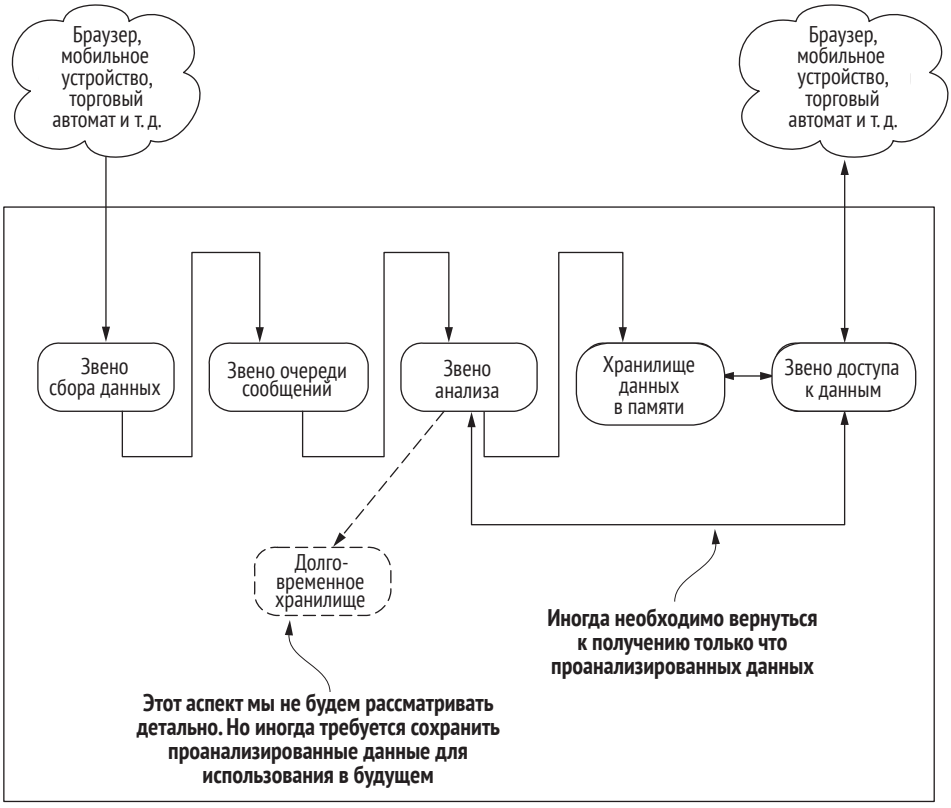
Архитектурная диаграмма потоковой обработки данных



Оглавление

Предисловие

Сколько себя помню, я всегда приходил в восторг от скорости обработки данных компьютером и стремился найти способ, как решить задачу быстрее. В конце 1990-х годов я в основном занимался программированием на C++ и моим любимым ключевым словом было `__asm`, означающее, что «следующий далее блок написан на языке ассемблера». Я понимал, что происходит на машинном уровне. В начале 2000-х я работал над программным обеспечением мобильных устройств, и тогда снова стал актуальным вопрос, как быстрее синхронизировать данные или ускорить работу устройств PalmPilot и Windows CE? В то время мы имели дело с гигантскими (по тогдашним меркам, конечно) медицинскими базами данных (объемом 25–50 МБ), для хранения которых в PalmPilot нужно было вставлять внешнюю карту памяти, и разрабатывали приложения, которые должны были обеспечить быстрый интерактивный поиск и просмотр этих данных.

По мере роста объемов данных в интересующих меня отраслях я оказался в точке, где большие наборы данных сталкиваются с необходимостью их быстрой обработки для понимания сути дела. Данные генерировались во все ускоряющемся темпе, а бизнес хотел все быстрее получать ответы на вопросы, связанные с этими данными. Как раз то, что мне было надо: большие данные и жажда скорости. Где-то в 2001 году я начал работать над приложениями для анализа рынка и электронной коммерции, когда данные непрерывно обновлялись, а нам нужно было извлекать из них смысл почти в режиме реального времени. В 2009 году я перешел в компанию Webtrends, где моя любовь к скорости и своевременной обработке данных достигла зрелости. Основным бизнесом Webtrends была аналитика, и наши идеи, касающиеся аналитики в реальном времени, как раз стали вызывать интерес у заказчиков. Первый проект, над которым я работал, заключался в том, чтобы выводить ключевые показатели на информационную панель с задержкой не более пяти минут относительно потока данных о событиях на сайте, где бы они ни происходили. В то время это выходило за рамки обыденности.

В 2011 году я вошел в группу разработки новых продуктов. Нашей задачей было и дальше продвигать идею аналитики в реальном времени и попытаться взорвать нашу отрасль изнутри. Мы потратили уйму времени на исследования, создание прототипа и обдумывание следующего шага, и тут случился идеальный шторм. Мы уже поглядывали в сторону Apache Kafka, когда в сентябре 2011 был открыт исходный код Apache Storm. Мы тут же набросились на него с бешеным энтузиазмом. Уже зимой мы проде-

монстрировали то, чем занимались, нескольким заинтересованным клиентам. В то время мы не оглядывались назад и всецело сосредоточились на выполнении соглашения об уровне услуг (Service Level Agreement, SLA), звучавшем примерно так: «от щелчка мышью в любой точке света до информационной панели – три секунды!». Спустя много месяцев труда куда большей по размеру команды мы выполнили свое обещание и нас признали технологией года в области цифровой аналитики (Digital Analytics New Technology of the Year – www.digitalanalyticsassociation.org/awards2013). Я был поглощен архитектурным проектированием и разработкой всех аспектов этого решения – от сбора данных до первого варианта пользовательского интерфейса (который ласково называл «голым скелетом», поскольку опыта в области построения UI у меня не было).

Мы не оставили свое увлечение и начали знакомиться с технологией Spark Streaming, когда она еще не вышла за стены Berkley AMPLab. С той поры я занимаюсь созданием все новых потоковых систем, стремясь к тому, чтобы полезная информация извлекалась из данных со скоростью мысли. Я выступаю на международных конференциях по этой тематике и работаю с компаниями по всему миру, помогая им проектировать и разрабатывать системы, решающие проблемы потоковой обработки данных.

Даже сегодня не все хорошо понимают, из каких частей состоит потоковая система. Найти информацию об отдельных компонентах нетрудно, но четкое понимание того, как устроен весь конвейер и как связаны между собой его звенья, встречается редко.

Поэтому я с огромным удовольствием попытался поделиться в этой книге своим практическим опытом и знаниями. Моя цель – предложить прочный фундамент, на котором можно строить и исследовать полноценные потоковые системы.

Благодарности

Прежде всего, хочу поблагодарить свою семью за поддержку во время работы над книгой. На протяжении многих выходных и вечеров я только и говорил: «Извините, я не могу помочь в саду (или поиграть в лакросс, или пойти в гости)». Конечно, и детям нелегко было это слышать, и жена устала все время брать на себя мою работу. Но на всем тернистом пути поддержка со стороны семьи ни разу не пошатнулась, она была для меня постоянным источником бодрости и вдохновения. Мой долг перед женой и детьми огромен, для его выражения одного «спасибо» недостаточно.

Спасибо Карен, моему редактору-консультанту, за бесконечное терпение, понимание и готовность в любое время обговаривать со мной различные вопросы. Спасибо Робину, рецензенту со стороны издательства, который поверил в меня, пестовал идею этой книги, и служил слушателем, благодаря которому поезд не сошел с рельс на ранних этапах пути. Спасибо Берту, который учил меня рассказывать историю, находить подходящий уровень глубины и педагогически правильно выстраивать техническую книгу. Спасибо техническому редактору Грегору, чьи глубокие и полезные замечания помогли сделать книгу такой, какой вы ее видите. И наконец, спасибо всему коллективу издательства Manning за фантастическую работу, благодаря которой мы все-таки дошли до конца.

Спасибо всем, кто купил и прочитал ранние варианты рукописи в рамках программы предварительного ознакомления, всем, кто оставлял сообщения на форуме автора, а также многочисленным рецензентам за их бесценные замечания, в частности: Эндрю Джибсону (Andrew Gibson), д-ру Тобиасу Бюргеру (Tobias Bürger), Джейку Маккрэри (Jake McCrary), Родриго Абрю (Rodrigo Abreu), Эндт Кеффаласу (Andy Keffalas), Джону Гатри (John Guthrie), Космасу Чатзимихалису (Kosmas Chatzimichalis), Джулиано Бертоти (Giuliano Bertoti), Карлосу Куротто (Carlos Curotto), Энди Киршу (Andy Kirsch), Дугласу Данкану (Douglas Duncan), Джеффу Смиту (Jeff Smith) и Серджио Фернандесу Гонсалесу (Sergio Fernández González), Яромиру Д. Б. Немецу (Jaromir D. B. Nemec), Хосе Самонте (Jose Samonte), Яну Ноннену (Jan Nonnen), Ромиту Сингхаи (Romit Singhai), Крису Аллену (Chris Allan), Джонатану Томсу (Jonathan Thoms), Стивену Дженкису (Steven Jenkins), Лее Джилберту (Lee Gilbert), Амандипу Хурана (Amandeep Khurana), Чарли Гейнсу (Charlie Gaines). Без вас эта книга не состоялась бы.

Еще многие принимали участие в создании книги тем или иным способом, но я не могу упомянуть всех поименно, потому что эдак благодарности никогда бы не закончились. Тем не менее, большое спасибо всем, кто помогал мне на этом пути!

Об этой книге

Системы реального времени появились давно, но в течение многих лет режим реального времени и потоковой обработки оставался вотчиной аппаратных систем. В таких системах невыполнение SLA угрожает человеческой жизни. Но за последние десять лет появились и стали быстро распространяться системы почти реального времени. Примеры потоковой обработки встречаются повсюду: социальные сети, игры, умные города, интеллектуальные измерительные устройства, ваша новая стиральная машина – список можно продолжать долго. Подумайте вот о чем: если считать, что байт равен одному галлону воды, то сегодня среднего размера дом можно было бы заполнить за 10 секунд, а к 2020 году это займет всего 2 секунды. Чтобы извлечь смысл из такого потопа данных, нужно потоковые системы.

У этой книги, посвященной идеям потоковой обработки данных в режиме реального времени, две цели. Первая – научить вас рассуждать о конвейере в целом, чтобы вы могли не только построить потоковую систему, но и понимали, на какие компромиссы приходится идти в каждом ее звене. Вторая – заложить фундамент, опираясь на который вы сможете глубже исследовать каждое звено, если это потребуется в интересах дела или просто ради удовлетворения любопытства.

Как работать с книгой?

Предполагалось, что книгу будут читать с начала до конца, но каждая глава содержит достаточно информации, чтобы ее можно было прочитать и понять отдельно от других. Поэтому, если вы захотите узнать о конкретном звене, можете перейти прямо к соответствующей главе, а затем воспользоваться полученной информацией в качестве основы для более глубокого изучения остальных глав.

Кому стоит прочитать эту книгу?

Эта книга рассчитана на разработчиков и архитекторов, но написана так, чтобы ее легко могли понять технические руководители и люди, принимающие решения о развитии бизнеса, – никакого предварительного знакомства с системами реального времени или потоковой обработки данных не предполагается. Единственное техническое требование – умение читать код, написанный на Java. На этом языке написаны как сами рассматриваемые системы, так и пример кода в главе 9.

Структура книги

Структура книги изображена на рис. 1.

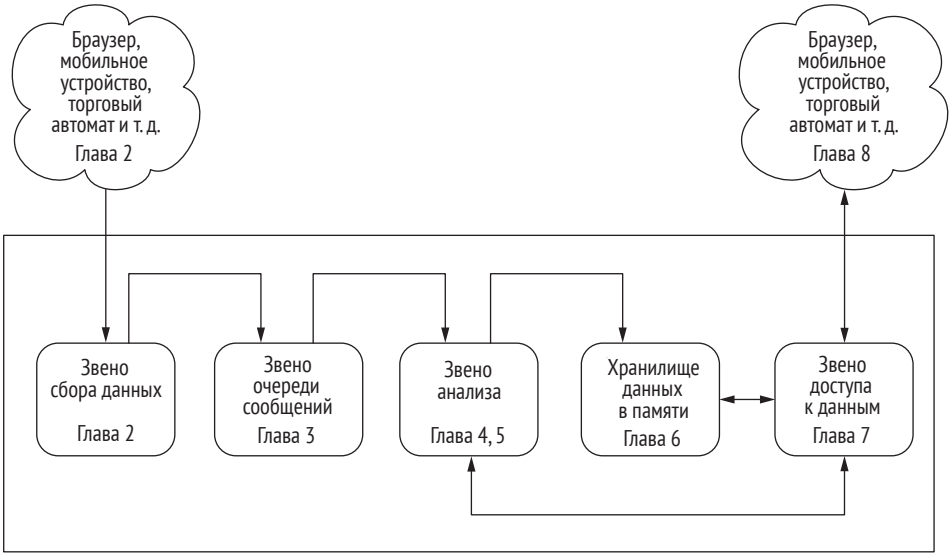


Рис. 1. Архитектурная диаграмма с разбиением на главы

Глава 1 содержит краткий план всей книги, он может послужить в качестве карты на случай, если понадобится найти нужное звено. Вслед за планом определяется, что такое система реального времени, в чем состоит различие между системой реального времени и своевременной системой, и кратко затрагивается вопрос о важности безопасности (которому можно посвятить отдельную книгу).

В главе 2 рассматриваются все аспекты сбора данных для потоковой системы: интерактивные средства, масштабирование и отказоустойчивость. Эта глава содержит все относящееся к звену сбора данных. Прочитав ее, вы будете готовы к созданию надежного и масштабируемого звена.

Глава 3 посвящена тому, как разорвать связь между собираемыми и анализируемыми данными, разместив между ними звено очереди сообщений. Вы узнаете, зачем нужно это звено, что такое долговечность сообщений, какие бывают семантики доставки сообщений и как выбрать технологию, подходящую для решения поставленной перед вами задачи.

В главе 4 мы займемся распространенными архитектурными паттернами распределенной потоковой обработки и рассмотрим, что означает семантика доставки сообщений в этом звене, как работать с состоянием, что такое отказоустойчивость и почему она необходима.

В главе 5 мы перейдем от обсуждения архитектуры к опросу потока, проблемам, касающимся времени, и четырем популярным техникам обобще-

ния. Если глава 4 отвечает на вопрос, *что* такое распределенная система потоковой обработки, то в главе 5 рассматривается ответ на вопрос, *как* она устроена.

В главе 6 обсуждаются варианты хранения данных в памяти в процессе анализа и после него. Мы не будем тратить время на обсуждение методов долгосрочного хранения данных на диске, потому что такие решения зачастую используются за рамками потокового анализа и не способны обеспечить такую же производительность, как хранилища в памяти.

В главе 7 мы начнем разговор о том, что делать с данными, которые мы собрали и проанализировали. Здесь обсуждаются варианты коммуникации и протоколы отправки данных потоковому клиенту. Попутно мы ответим на вопрос, как бизнес-требования соотносятся с различными протоколами и как выбрать подходящий протокол.

В главе 8 речь идет о концепциях, которые следует иметь в виду при построении потокового клиента. Эта глава не просто о том, как разработать веб-приложение на основе HTML, в ней рассматриваются гораздо более глубокие, низкоуровневые вопросы проектирования клиентской части потоковой системы.

Глава 9... что ж, если вы прочитали все, что ей предшествует, принимайте поздравления! В первых восьми главах было рассмотрено много материала. А в главе 9 мы претворим теорию в практику – построим полный конвейер потоковой обработки данных и обсудим, как сделать из демонстрационного примера производственное приложение.

О коде

Весь код, представленный в последней главе, можно найти в сети. Код можно скачать бесплатно с сайта издательства Manning по адресу `www.manning.com/books/streaming-data` или найти его на GitHub по адресу `https://github.com/apsaltis/StreamingData-Book-Examples`.

Код разбит на отдельные Maven-проекты, по одному для каждого рассматриваемого в главе 9 звена. Инструкции по сборке и запуску программ приводятся по ходу дела.

Исходный код в листингах и внутри текста набран моноширинным шрифтом. Некоторые листинги аннотированы, чтобы пояснить ключевые моменты.

Об авторе

Эндрю Пселтис одержим потоковыми системами, и все его устремления направлены на то, чтобы извлекать смысл из данных со скоростью мысли. Большую часть своего времени (когда не спит) он размышляет о потоковых системах, пишет о них или разрабатывает их. Он помогает клиентам всех мастей строить или исправлять сложные потоковые системы, расска-

зывает о потоковой обработке на всех континентах и учит других создавать потоковые системы. Время, остающееся от трудов, он посвящает прелестной жене, двоим ребятишкам и игре в лакросс – как болельщик.

Автор в сети

Приобретение книги «Потоковая обработка данных» открывает бесплатный доступ к закрытому форуму, организованному издательством Manning Publications, где вы можете оставить свои комментарии к книге, задать технические вопросы и получить помощь от автора и других пользователей. Получить доступ к форуму и подписаться на список рассылки можно на странице www.manning.com/books/streaming-data. Там же написано, как зайти на форум после регистрации, на какую помощь можно рассчитывать, и изложены правила поведения в форуме.

Издательство Manning обязуется предоставлять читателям площадку для общения с другими читателями и автором. Однако это не означает, что автор обязан как-то участвовать в обсуждениях; его присутствие на форуме остается чисто добровольным (и не оплачивается). Мы советуем задавать автору какие-нибудь хитроумные вопросы, чтобы его интерес к форуму не угасал!

Форум автора в сети и архивы будут доступны на сайте издательства до тех пор, пока книга не перестанет печататься.

Об иллюстрации на обложке

Рисунок на обложке книги называется «Традиционный костюм марокканца, зима 1695 года». Рисунок взят из четырехтомного сочинения Томаса Джеффри «Коллекция одежды различных народов от древних времен до наших дней», изданного в Лондоне между 1757 и 1772 годами. На титульной странице говорится, что это гравюры на меди, раскрашенные вручную с добавлением гуммиарабика. Томаса Джеффри (1719–1771) называли «географом короля Георга III». В свое время он был главным в Англии поставщиком карт. Он гравировал и печатал карты для правительства и других официальных учреждений, а также изготавливал самые разные коммерческие карты и атласы, особенно Северной Америки. Работы в качестве картографа возбудила в нем интерес к традиционным костюмам в тех местах, которые он наносил на карту. Эти костюмы блистательно изображены в его коллекции.

Увлечение дальними землями и путешествиями ради удовольствия в конце XVIII века было еще относительно новым явлением, и такие коллекции были весьма популярны, поскольку давали как туристам, так и любителям странствовать, не вылезая из кресла, возможность познакомиться с обитателями других стран. Разнообразие иллюстраций в четырехтомнике

Джеффри красноречиво свидетельствует об уникальности и индивидуальности народов мира, которое можно было наблюдать каких-то 200 лет назад. С тех пор манера одеваться сильно изменилась, и различия между областями, когда-то столь разительные, сгладились. Теперь трудно отличить друг от друга даже выходцев с разных континентов, что уж говорить о странах или областях. Но можно взглянуть на это и с оптимизмом – мы обменяли культурное и визуальное разнообразие на иное устройство личной жизни – основанное на многостороннем и стремительном технологическом и интеллектуальном развитии.

Во времена, когда трудно отличить одну компьютерную книгу от другой, издательство Manning откликается на новации и инициативы в компьютерной отрасли обложками своих книг, на которых представлено широкое разнообразие местных укладов быта в позапрошлом веке. Мы возвращаем его в том виде, в каком оно запечатлено на рисунках из собрания Джеффри.

Часть 1

Новый целостный подход

Данные окружают нас повсюду, и каждый день в сети появляются новые источники данных. Если вы пока не сталкивались с созданием систем обработки данных в реальном времени, то столкнетесь в ближайшем будущем. Само существование все большего числа предприятий зависит от умения обрабатывать потоки данных и принимать решения в зависимости от результатов. В первой части книги мы рассмотрим жизненный цикл потоковой системы с момента попадания в нее данных и до момента их отображения или потребления другими системами.

В главе 1 мы познакомимся с идеей потоковой обработки данных и с применяемой терминологией. Для разных людей слова «потоковая обработка данных» и «реальное время» могут означать разные вещи. Поэтому мы уточним, что понимаем под этими терминами *мы*, и приведем архитектурную диаграмму, которую будем наполнять содержанием на протяжении всей книги. В конце главы 1 мы скажем несколько слов о безопасности в контексте потоковых систем.

Точкой входа в потоковую систему является сбор данных. Вопрос о том, как собирать данные и предотвращать их потерю, находится в центре главы 2.

Собранные данные нужно как можно быстрее поместить в очередь сообщений (иногда ее называют *буфером* сообщений). Применяемая в этом звене технология характеризуется различными уровнями долговечности, семантикой доставки и влиянием на производителей и потребителей данных. В главе 3 мы расскажем, как учитывать эти особенности, и поделимся некоторыми рекомендациями.

Глава 4 посвящена анализу потоковых данных. Основное внимание уделяется анализу данных в темпе поступления, распространенным архитектурам потоковой обработки и ключевым функциям, общим для всех движков распределенной потоковой обработки.

При работе с распределенной системой потоковой обработки нужно учитывать много нюансов. Например, время. Как следует представлять себе время в потоковой системе? Ответ на этот вопрос дается в главе 5. Здесь же рассматриваются четыре общеупотребительных метода обобщения, применяемых в процессе анализа потоковых данных.

После анализа поток данных иногда требуется сохранить. Звучит странно – зачем сохранять поток, раз мы его обрабатываем! В главе 6 объясняется, почему возникает необходимость в сохранении данных, что именно следует сохранять и как это правильно сделать в процессе обработки потока.

Когда мы доберемся до главы 7, данные уже собраны, проанализированы и, возможно, сохранены. И следующий шаг – сделать данные доступными, потому что в конечном итоге мы хотим передать результаты анализа другой системе, которая сможет предпринять какие-то действия на основе потоковых данных.

В главе 8 мы подведем итог первой части, обсудив основные принципы построения потокового клиента. Мы познакомимся с веб-клиентом и в заключение поговорим об опросе потока данных, поскольку пользователям это бывает нужно.

Глава 1

Введение в потоковую обработку данных

Краткое содержание главы:

- различие между системами реального времени и системами потоковой обработки данных;
- почему потоковая обработка данных важна;
- архитектурная диаграмма;
- безопасность в системах потоковой обработки данных.

Данные окружают нас повсеместно: благодаря телефонам, кредитным картам, оборудованным датчиками зданиям, торговым автоматам, термостатам, поездам, автобусам, самолетам, сообщениям в социальных сетях, цифровым фотографиям и видео – список можно продолжать. В отчете, датированным маем 2013 года, скандинавский исследовательский центр Sintef дал оценку, согласно которой приблизительно 90 % данных, существовавших на тот день в мире, были сгенерированы в течение двух предшествующих лет. В апреле 2014 года EMC вместе с IDC выпустили седьмое ежегодное исследование цифровой вселенной (www.emc.com/about/news/press/2014/20140409-01.htm), в котором утверждалось, что цифровая вселенная удваивается каждые два года, и между 2013 и 2020 годом ее размер возрастет десятикратно – с 4,4 до 44 триллионов гигабайт. Не знаю, как вы, а я с трудом могу переварить эти цифры и соотнести их с чем-нибудь в реальном мире. Хорошее сравнение приведено в том же отчете: если считать что один байт – это галлон воды, то для заполнения дома среднего размера в 2013 году было достаточно 10 секунд. А в 2020 хватит и двух секунд.

Понятие «больших данных» существует уже довольно давно, но теперь у нас есть технология, позволяющая сохранить и проанализировать все собранные данные. Это не устраняет потребность использовать данные

в подходящем контексте, но стало гораздо проще задавать интересные вопросы о данных, быстрее принимать более обоснованные решения и предлагать службы, позволяющие потребителям и предприятиям использовать данные о том, что происходит вокруг.

Мы живем в мире, который все сильнее ориентирован на *сейчас*: это и социальные сети, и розничные магазины, отслеживающие перемещение покупателей по проходам, и датчики, реагирующие на изменения в окружающей среде. Нет недостатка в примерах использования данных в момент порождения. Но вот чего не хватает, так это единого способа рассуждать о таких системах – и проектировать их – который учитывал бы не только имеющиеся, но и будущие службы.

В этой книге представлены общие архитектурные принципы, позволяющие обсуждать и проектировать системы, способные ответить на все поразительные вопросы об окружающих нас данных, которые еще только предстоит задать. Даже если вы никогда не проектировали, не разрабатывали и не работали с системами реального времени или системами для обработки больших данных, эта книга все равно послужит полезным руководством. Она посвящена важным идеям потоковой обработки данных в режиме реального времени. Но никакого предварительного опыта в этой области не предполагается, так что книга будет отличным подспорьем для разработчиков и архитекторов, желающих побольше узнать о таких системах. Она написана так, что будет полезной также техническим руководителям и лицам, принимающим решения о развитии бизнеса.

Чтобы подготовить почву, мы в этой главе познакомимся с концепцией систем потоковой обработки данных и с архитектурной диаграммой в предвкушении более глубокого изучения каждого звена в последующих главах. Но первым делом нужно понять, что вообще такое системы реального времени и потоковой обработки.

1.1. Что такое система реального времени?

Системы реального времени и *вычисления в режиме реального времени* существуют уже несколько десятков лет, но с пришествием Интернета их популярность резко возросла. Увы, вместе с популярностью пришли сомнения и споры. Из чего же состоит система реального времени?

Есть три типа систем реального времени: *жесткого реального времени*, *мягкого реального времени* и *почти реального времени*. Определения жесткого и мягкого реального времени взяты мной из книги Hermann Kopetz «Real-Time Systems» (Springer, 2011), а определение почти реального времени – со страницы `http://c2.com/cgi/wiki?NearRealTime`. За примером неоднозначности не надо ходить дальше определения на сайте Dictionary.com: «Обозначает или относится к системе обработки данных, работающей немного медленнее, чем система реального времени». Чтобы разобраться,

в чем тут дело, в табл. 1.1 приведена широкоупотребительная классификация систем реального времени вместе с основными характеристиками, по которым они различаются.

Таблица 1.1. Классификация систем реального времени

Классификация	Примеры	Единица измерения задержки	Терпимость к запаздыванию
Жесткая	Кардиостимулятор, антиблокировочная тормозная система	Микросекунды – миллисекунды	Нулевая – полный отказ системы, потенциальная смерть человека
Мягкая	Система резервирования авиабилетов, электронные биржевые торги, VoIP (Skype)	Миллисекунды – секунды	Низкая – без отказа системы, без угрозы жизни
Почти	Видео в Skype, домашняя автоматика	Секунды – минуты	Высокая – без отказа системы, без угрозы жизни

Опознать систему жесткого реального времени довольно просто. Почти всегда они встречаются во встраиваемых системах и характеризуются очень строгими временными ограничениями, невыполнение которых может привести к полному отказу системы. Вопрос о проектировании и реализации систем жесткого реального времени хорошо изучен в литературе, но выходит за рамки этой книги (интересующихся читателей отправляю к вышеупомянутой книге Германа Копеца).

Решить, идет ли речь о системе мягкого или почти реального времени, – интересное упражнение, поскольку частично перекрывающиеся определения создают почву для недоразумений. Приведем три примера.

- Человек, на твиттер которого вы подписаны, отправляет сообщение, и через несколько мгновений вы видите его в своем клиенте.
- Вы следите за самолетами в окрестности Нью-Йорка, пользуясь службой Live Flight Tracking, предоставляемой компанией FlightAware (http://flightaware.com/live/airport/KJFK).
- Вы пользуетесь приложением NASDAQ Real Time Quotes (www.nasdaq.com/quotes/real-time.aspx) для наблюдения за котировками своих любимых акций в режиме реального времени.

Хотя это совершенно разные системы, на рис. 1.1 показано, что между ними общего.

Во всех трех примерах разумно предположить, что допустимая временная задержка – всего несколько секунд, что угрозы жизни нет и что случайное запаздывание на несколько минут не приведет к полному отказу системы. Так? Если вы видите отправленный твит почти сразу, это мягкое или почти реальное время? А как насчет наблюдения за состоянием рейса или за котировками акций в режиме реального времени? В обоих случаях

возможно запаздывание – например, из-за медленной сети Wi-Fi в кофейне или на борту самолета. На этих примерах видно, что граница между системами мягкого и почти реального времени размыта, иногда исчезает вовсе, очень субъективна и часто зависит от потребителя данных.

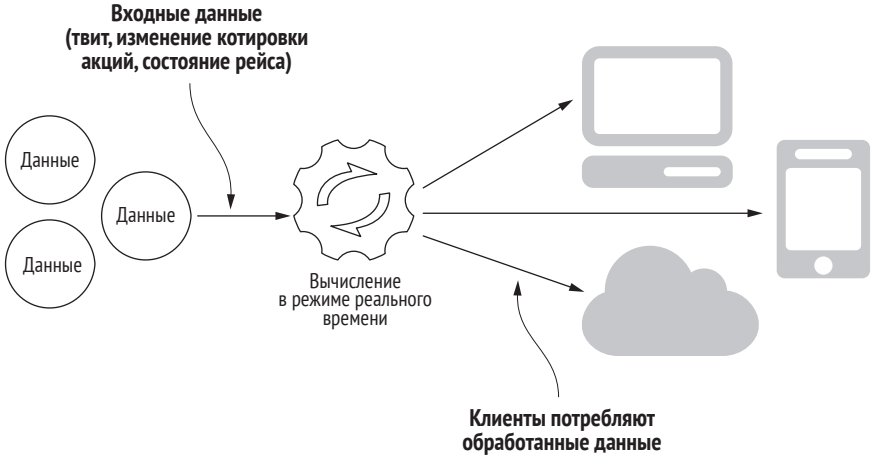


Рис. 1.1. Типичная система реального времени с потребителями

А теперь исключим из рассмотрения потребителя и сосредоточимся на самих службах.

- Твит отправляется в Твиттер.
- Служба Live Flight Tracking от компании FlightAware отслеживает авиарейсы.
- Приложение NASDAQ Real Time Quotes отслеживает котировки акций.

Мы, конечно, не знаем, как эти системы устроены внутри, но относительно любой из них можем задать один и тот же вопрос:

> *Является ли процесс доставки данных до точки, в которой их можно потребить, процессом мягкого или почти реального времени?*

Графически это изображено на рис. 1.2.

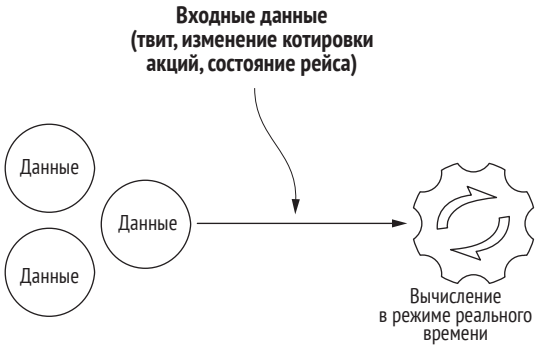


Рис. 1.2. Типичная система реального времени без потребителей

Если абстрагироваться от потребителей данных и сосредоточиться только на их обработке, то ответ изменится? Например, как бы вы классифицировали следующие процессы:

- твит отправлен в Твиттер;
- твит отправлен человеком, за которым вы наблюдаете в своем твиттер-клиенте.

Если вы классифицировали их по-разному, то почему? Из-за задержки или воспринимаемой задержки появления твита в клиенте? Проходит некоторое время – и граница между системой мягкого и почти реального времени совсем размывается. Зачастую пользователи называют их просто системами реального времени. В этой книге я собираюсь использовать более правильный способ идентификации таких систем.

1.2. Различия между системами реального времени и потоковыми системами

Сейчас уже должно быть ясно, что система считается системой мягкого или почти реального времени в зависимости от того, как воспринимают задержку пользователи. На простых примерах мы видели, что провести различие между этими типами систем реального времени не всегда просто. А если в оценке системы принимает участие несколько людей, то проблема может еще обостриться. Наша цель – выработать общий язык, на котором можно было бы описывать такие системы. Взглянув на проблему в целом, мы приходим к выводу, что пытаемся использовать один термин для определения двух частей большой системы. На рис. 1.3 показано, что такой подход обречен на неудачу: очень трудно разговаривать о системе с другими людьми, потому что у нас нет четкого определения.

На левом рисунке мы видим службу нежесткого реального времени, или *вычислительную* часть системы, а на правом – клиентов, или *потребляющую* часть системы.

Определение: система потоковой обработки данных. Во многих ситуациях вычислительная часть системы работает в режиме нежесткого реального времени, но клиенты потребляют данные не в реальном времени из-за сетевых задержек, дизайна приложения или просто потому, что клиентское приложение не запущено. Иначе говоря, мы имеем службу нежесткого реального времени и клиентов, которые потребляют данные, когда захотят. Это и называется *системой потоковой обработки данных* – система нежесткого реального времени, которая делает данные доступными, когда клиентское приложение хочет их видеть. Это не система мягкого или почти реального времени, это потоковая система.

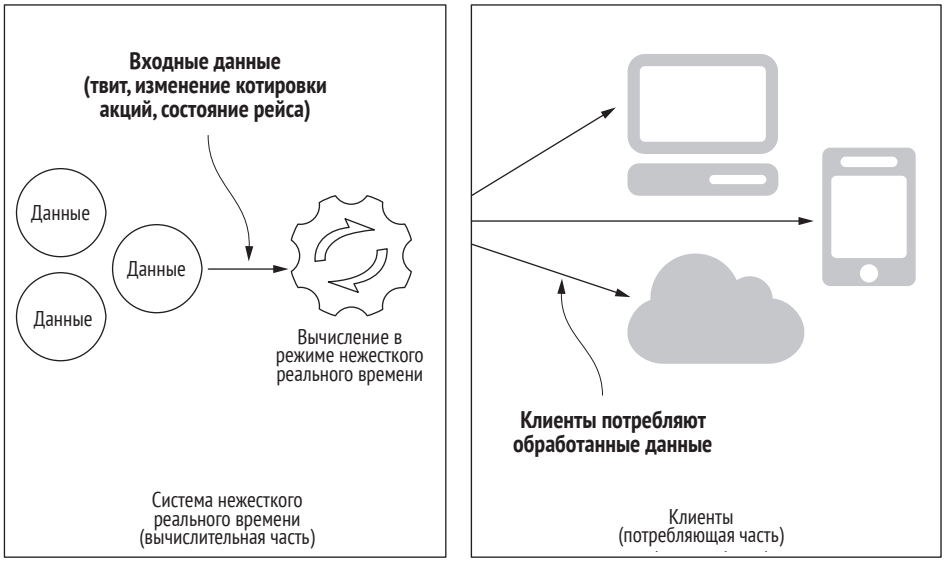


Рис. 1.3. Разделение вычисления в реальном времени и потребления

На рис. 1.4 показано, как это определение применяется к архитектуре на рис. 1.3.

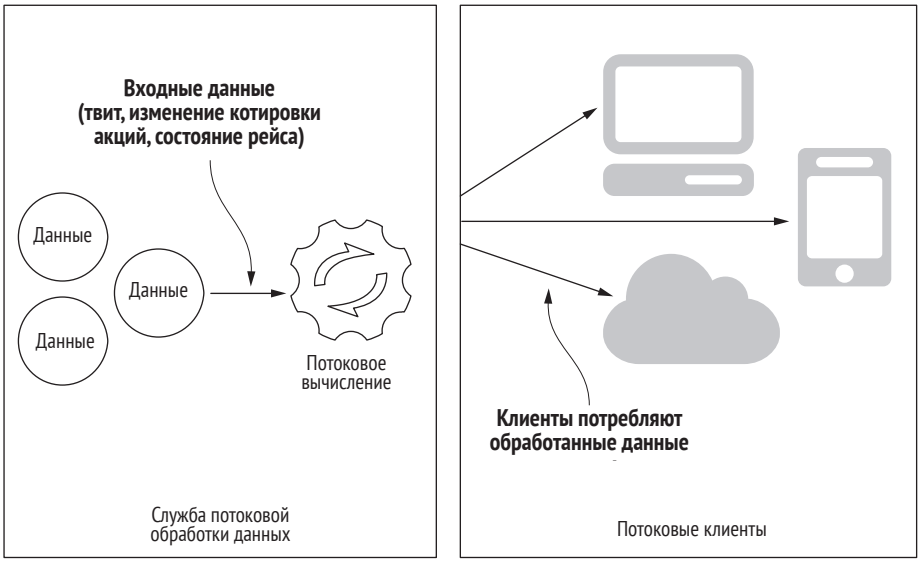


Рис. 1.4. Первый взгляд на систему потоковой обработки данных

Концепция потоковой обработки данных исключает всякие недоразумения, связанные с нечетким различием между системами мягкого реального времени, почти реального времени и вообще не реального времени, и позволяет сосредоточиться на проектировании систем, которые достав-

ляют информацию в тот момент, когда клиент ее запрашивает. Рассмотрим приведенные выше примеры с точки зрения потоковой обработки. Сможете ли вы в каждом случае отделить потоковую службу от потокового клиента?

Человек, на твиттер которого вы подписаны, отправляет сообщение, и через несколько мгновений вы видите его в своем клиенте.

Вы следите за самолетами в окрестности Нью-Йорка, пользуясь службой Live Flight Tracking, предоставляемой компанией FlightAware.

Вы пользуетесь приложением NASDAQ Real Time Quotes для наблюдения за котировками своих любимых акций в режиме реального времени.

Ну и что у вас получилось? Лично я думаю так.

- *Твиттер* – потоковая система, которая обрабатывает твиты и позволяет клиентам запрашивать последние твиты в тот момент, когда они нужны; иногда спустя несколько секунд после момента отправки, а иногда – через несколько часов.
- *FlightAware* – потоковая система, которая обрабатывает актуальную информацию о состоянии рейсов и позволяет клиентам запрашивать данные по конкретным аэропортам или рейсам.
- *NASDAQ Real Time Quotes* – потоковая система, которая обрабатывает котировки всех акций и позволяет клиентам запрашивать последнюю котировку определенных акций.

Вы обратили внимание, что теперь нет нужды задумываться о классификации системы реального времени? Важно лишь, какие данные и каким образом служба предоставляет своим клиентам, когда они посылают запрос. А раз так, то мы можем назвать такую систему *своевременной* – доставляющей данные в тот момент, когда они необходимы. Мы, конечно, не знаем, как эти системы в действительности работают, но ничего страшного. Мы научимся собирать подобные системы из компонентов с открытым исходным кодом, которые предназначены для потребления, обработки и представления потоков данных.

1.3. Архитектурная диаграмма

Разобравшись, что такое системы реального времени и потоковые системы, мы можем перейти к архитектурной диаграмме, которой будем пользоваться на протяжении всей книги. Она позволит рассуждать обо всех потоковых системах единообразным способом – на нашем языке паттернов. Эта диаграмма изображена на рис. 1.5. Познакомьтесь с ним.

По ходу изложения мы рассмотрим все звенья, показанные на рис. 1.5, не упуская из виду картину в целом. Хотя на диаграмме у каждого звена есть название, следует иметь в виду, что звенья определены не так строго и непоколебимо, как в некоторых других архитектурах. И хотя мы называ-

ем их звеньями, используются они скорее, как детали конструктора LEGO, что дает возможность спроектировать решение, отвечающее конкретной задаче. Наши звенья не подразумевают какого-то определенного сценария развертывания. Во многих случаях они даже физически находятся в различных местах.

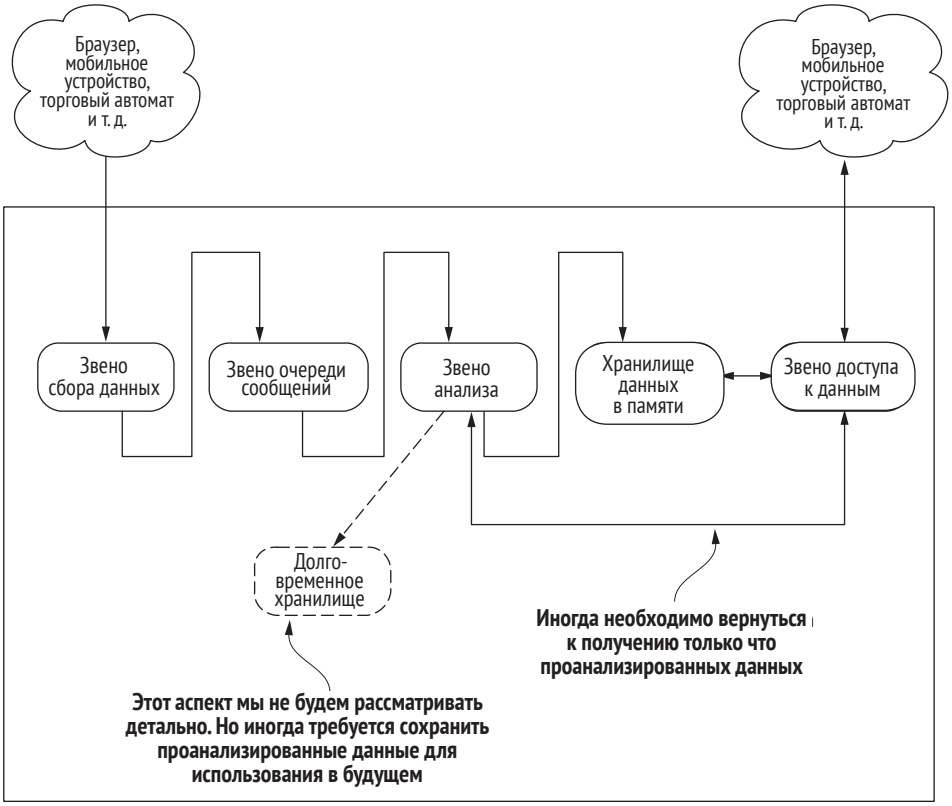


Рис. 1.5. Архитектурная диаграмма потоковой обработки данных

Обратимся к приведенным выше примерам. Посмотрим, как служба Твиттера ложится на нашу архитектуру.

- *Звено сбора данных*. Отправленные пользователями твиты собираются службами Твиттера.
- *Звено очереди сообщений*. Нет сомнений, что Твиттер задействует центры обработки данных в разных местах, и вероятно, сбор твитов производится не там же, где анализ.
- *Звено анализа*. Я уверен, что эти 140 символов подвергаются самым разным способам обработки, но уж, как минимум, Твиттер должен определить, кто подписан на данный твит.
- *Звено долгосрочного хранения*. Хотя мы и не собираемся обсуждать этот необязательный слой, нетрудно догадаться, что раз можно уви-

деть старые твиты, значит, они хранятся в каком-то постоянном хранилище.

- *Звено хранилища данных в памяти*. Твиты, отправленные в течение нескольких последних секунд, скорее всего, хранятся в памяти.
- *Доступ к данным*. Любой твиттер-клиент должен подключаться к Твиттеру, чтобы получить доступ к службе.

Самостоятельно проведите такой же анализ для двух других примеров.

- *FlightAware* – потоковая система, которая обрабатывает актуальную информацию о состоянии рейсов и позволяет клиентам запрашивать данные по конкретным аэропортам или рейсам.
- *NASDAQ Real Time Quotes* – потоковая система, которая обрабатывает котировки всех акций и позволяет клиентам запрашивать последнюю котировку определенных акций.

Получилось? Не переживайте, если задача показалась трудной или непривычной. В последующих главах примеров будет предостаточно. Мы проработаем их вместе, обращая внимание на каждое звено, и тогда станет понятно, как из этих деталей LEGO можно собрать решения различных практических задач.

1.4. Безопасность в контексте потоковых систем

На нашей архитектурной диаграмме вы не найдете явных упоминаний безопасности. Во многих случаях безопасность важна, но ее можно естественно наложить на эту архитектуру, как показано на рис. 1.6.

Мы не будем тратить время на детальное обсуждение вопросов безопасности, но по ходу дела я иногда буду вспоминать об этом, чтобы вы видели место безопасности в общей картине и могли поразмыслить над тем, что это может значить для решаемых вами задач. Если вас интересует углубленное рассмотрение безопасности в распределенных системах, обратитесь к книге Ross Anderson «Security Engineering: A Guide to Building Dependable Distributed Systems» (Wiley, 2008). Ее можно почитать бесплатно по адресу www.cl.cam.ac.uk/~rja14/book.html.

1.5. Как производится масштабирование?

На самом верхнем уровне есть два способа масштабирования: вертикальное и горизонтальное.

Под *вертикальным* масштабированием понимается наращивание мощности существующего оборудования (физического или виртуального) или программного обеспечения путем добавления ресурсов. При входе в ресторан иногда можно увидеть табличку, на которой указано максимальное

число мест. Когда приходят гости, можно накрыть дополнительные столы и принести еще стулья – это и есть вертикальное масштабирование. Но если заняты все места, то новых гостей усадить некуда. В конце концов, вместимость ограничена размером ресторана. В мире компьютеров примерами вертикального масштабирования могут служить оснащение сервера дополнительной памятью, процессорами или жесткими дисками. Но, как и в случае ресторана, мы ограничены максимальной емкостью системы – физической или виртуальной.

Рис. 1.6. Архитектурная диаграмма с выделением безопасности

Горизонтальное масштабирование знаменует другой подход к проблеме. Вместо того чтобы наращивать ресурсы сервера, мы увеличиваем количество серверов. Удачным примером горизонтального масштабирования может служить автострада. Возьмем двухполосную автостраду, первоначально рассчитанную на 2000 машин в час. Со временем вдоль автострады вырастают новые дома и коммерческие заведения, в результате чего поток машин возрастает до 8000 в час. Легко представить (и, возможно, вы знакомы с этим на собственном опыте), что случится: огромные пробки в часы пик и малоприятная езда в город и обратно в любое время. Для решения проблемы строятся дополнительные полосы – в итоге автострада масштабирована по горизонтали и может справиться с возросшим трафиком. Но еще эффективнее было бы, если бы автострада могла расширяться (добавление полос) и сужаться (удаление полос)

в зависимости от текущего трафика. На пунктах пропуска в аэропорт, когда пассажиропоток мал, оператор закрывает часть линий досмотра, а когда поток увеличивается, открывает. Если ваша служба размещена на серверах облачного провайдера (Google, AWS, Microsoft Azure), то вы можете воспользоваться такой эластичностью – она иногда называется *автомасштабированием*. Идея в том, что в периоды повышенного спроса на службу автоматически добавляются новые серверы, а когда спрос падает, серверы выводятся.

Современные способы проектирования системы ориентированы больше на горизонтальное масштабирование, но это не значит, что вертикальное масштабирование пора отправлять на свалку. Вертикальное масштабирование часто применяется для определения идеальной конфигурации ресурсов для службы, а затем служба масштабируется по горизонтали. Но в этой книге мы будем иметь дело в основном с горизонтальным масштабированием.

1.6. Резюме

Итак, мы познакомились с архитектурной диаграммой, и можно подвести некоторые итоги тому, что мы узнали.

- Мы определили, что такое система реального времени.
- Мы изучили различия между системами реального времени и потоковыми (своевременными) системами.
- Мы поняли, почему потоковая обработка важна.
- Мы нарисовали архитектурную диаграмму.
- Мы обсудили место безопасности в потоковых системах.

Не расстраивайтесь, если все это пока выглядит смутно или если задача применения архитектурной диаграммы к конкретным практическим проблемам кажется неподъемной. В последующих главах мы, не торопясь, рассмотрим много разных примеров. И к концу книги все эти идеи покажутся гораздо более естественными.

А сейчас мы готовы детально рассмотреть звенья, понять, из чего они состоят и как их применять к построению системы потоковой обработки данных. С какого звена начать? Взгляните на слегка модифицированную архитектурную диаграмму на рис. 1.7.

Мы будем рассматривать звенья по одному, слева направо, так что начнем со звена сбора данных.

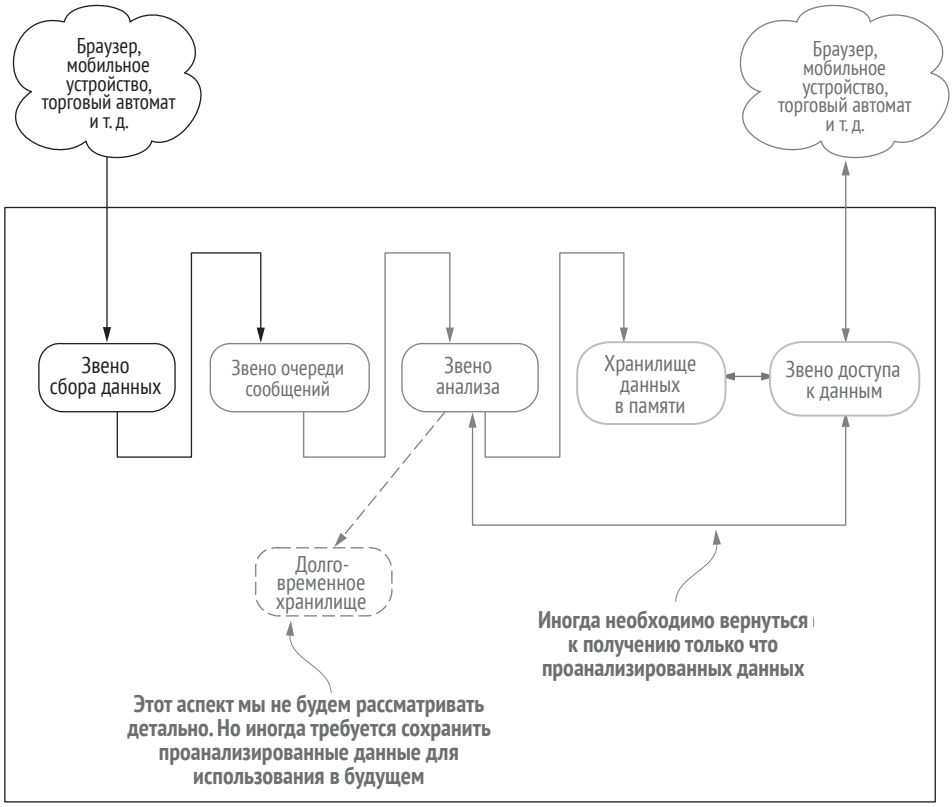


Рис. 1.7. Архитектурная диаграмма с упором на первое звено