Arenadata™ Streaming

Версия - v1.0.0

Общий обзор Arenadata Streaming

2019 Arenadata™

Оглавление

1 Концепция хранения **5**

2 Гарантии **8**

3 Рекомендации по использованию **9**
3.1 ADS как Messaging System 9
3.2 ADS как Storage System 10
3.3 ADS – Stream Processing 10

4 Концепция потоков NiFi **11**

Arenadata Streaming (**ADS**) – **платформа распределенных потоковых операций**, включающая интегрированный набор компонентов корпоративного уровня на базе решений с открытым исходным кодом. Платформа содержит в себе все необходимые компоненты для потоковой передачи и обработки данных в реальном времени, их преобразования, взаимодействия и хранения, передачи в семантике “exactly-once delivery”, интеграции, безопасности и администрирования. Также платформа может выступать в качестве корпоративной шины данных и ETL-инструмента.

Идея платформы распределенных потоковых операций заключается в обеспечении:

- Единой точки доступа – возможность использования в качестве корпоративной шины обмена данными для приложений;
- Легкого, безопасного и надежного способа управления потоком данных – возможность безопасного сбора больших потоков данных и эффективного управления ими в режиме реального времени;
- Политики безопасности – возможность создания потоков данных с поддержкой разграничения прав доступа к ним;
- Быстрой и непрерывной разработки – возможность разработки потоковых аналитических приложений за считанные минуты в режиме реального времени без единой строчки кода.

Одной из особенностей реализации платформы является применение техники, сходной с журналами транзакций, используемыми в системах управления базами данных. **ADS** обладает следующими отличительными техническими качествами:

- Отказоустойчивость;
- Масштабируемость;
- Распределенность;
- Доступное оборудование;
- Реальное время;
- Безопасность;
- Интеграция;
- Простота и гибкость.

Arenadata Streaming (**ADS**) – это полноценный дистрибутив платформы потоковых операций на базе **Apache Kafka**, адаптированный для корпоративного использования. **Apache Kafka** – распределённый программный брокер сообщений, проект с открытым исходным кодом, разработанный в рамках **Apache Software Foundation**.

Текущий релиз версии **ADS 1.0.0** вышел во втором квартале 2018 года. В состав версии входят следующие компоненты: **Ambari**, **Zookeeper**, **NiFi**, **Kafka**. Интеграция с **ZooKeeper** позволяет системе работать не только быстро и слаженно, но безопасно, что особенно важно в случае больших данных.

Платформа потоковой передачи данных имеет три ключевые возможности:

- Публикация и подписка на потоковую передачу данных, похожую на очередь сообщений или корпоративную систему обмена сообщениями;
- Хранение потоков записей отказоустойчивым способом;
- Обработка потоков записей по мере их возникновения.

Платформа **ADS**, как правило, используется для двух обширных классов приложений:

- Создание канала для потоковой передачи данных в реальном времени с целью надежного обмена данными между системами и приложениями;

- Создание приложений для потоковой передачи данных в реальном времени с целью их преобразования и взаимодействия с другими потоками.

Несколько концепций **Arenadata Streaming**:

- **ADS** запускается как кластер на одном или нескольких серверах, которые могут располагаться в разных центрах обработки данных;
- В кластере **ADS** потоки записей хранятся по категориям, называемым *topics* (топики);
- Каждая запись состоит из ключа, значения и временной метки.

ADS имеет четыре основных **API** (Рис.1.):

- **Producer API** (поставщик) позволяет приложению публиковать поток записей по одному или нескольким топикам платформы. Примеры использования приведены в javadocs;
- **Consumer API** (потребитель) позволяет приложению подписываться на один или несколько топиков и обрабатывать принадлежащие им потоки записей. Примеры использования приведены в javadocs;
- **Streams API** позволяет приложению выступать в качестве *stream processor* (потокового процессора), потребляя входной поток данных из одного или нескольких топиков и создавая выходной поток данных так же для одного или нескольких топиков, эффективно преобразуя входные потоки в выходные. Примеры использования приведены в javadocs;
- **Connector API** позволяет создавать и запускать повторное использование поставщиков и потребителей, которые связывают топики с существующими приложениями или системами данных. Например, коннектор для реляционной базы данных может записывать каждое изменение в таблицу. Примеры использования приведены в javadocs.

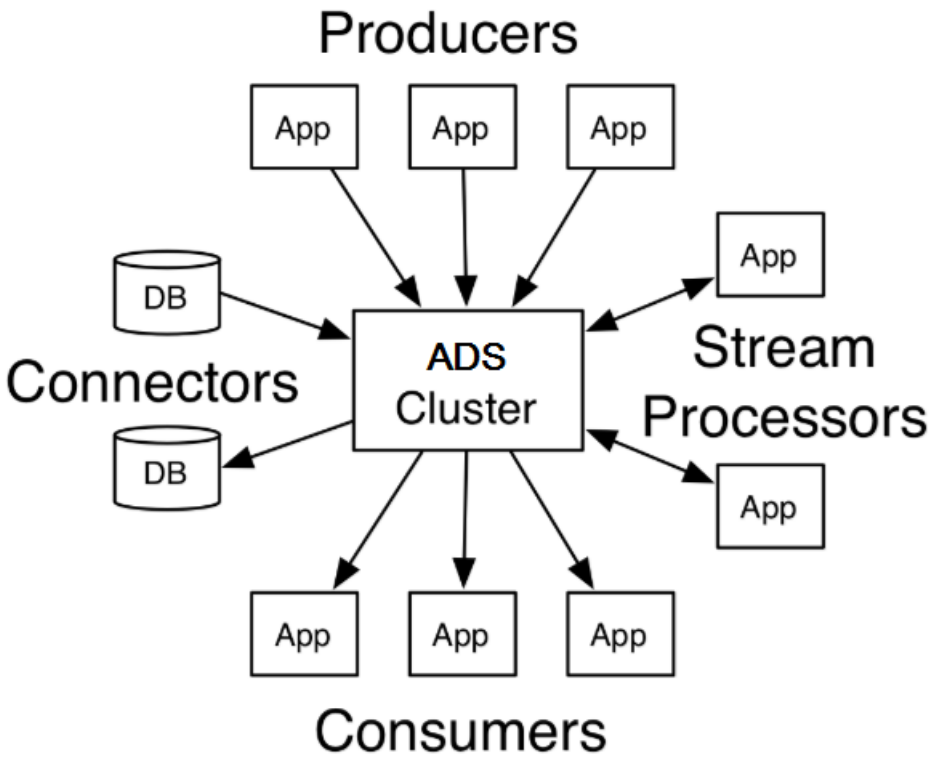


Рис.1.: API платформы ADS

Клиенты для работы с **Apache Kafka** доступны на многих языках программирования (Clients).

Сервис **NiFi** в составе платформы **ADS** мощный инструмент для построения масштабируемых ориентированных графов маршрутизации данных и их преобразования. Некоторые из высокоуровневых возможностей и целей **NiFi**:

- Веб-интерфейс пользователя
  - Разработка, управление и мониторинг в едином интерфейсе
- Гибкое конфигурирование в зависимости от потребностей
  - Устойчивость к потерям или гарантированная доставка
  - Низкая задержка или высокая пропускной способности
  - Динамическое определение приоритетов
  - Возможность изменения потока во время выполнения
- Происхождение данных
  - Отслеживание потока данных от начала до конца
- Расширение функционциональных возможностей
  - Возможность создания собственных процессоров и многого другого
  - Обеспечение быстрой разработки и эффективного тестирования
- Безопасность
  - SSL, SSH, HTTPS, зашифрованный контент и т.д.
  - Multi-tenant авторизация и внутренняя авторизация/управление политикой

В документации приведены концепции хранения в платформе **Arenadata Streaming**, гарантии и рекомендации по использованию **ADS**. Раздел предлагается к прочтению перед переходом к непосредственной установке системы.

Important: Контактная информация службы поддержки – e-mail: info@arenadata.io

Глава 1

Концепция хранения

Для начала в главе описывается основная абстракция, которую **ADS** обеспечивает для потока записей – топик.

Топики – это категории, по которым записи публикуются в платформе. В **ADS** топики могут иметь нескольких потребителей, которые подписываются на получение находящихся в них данных. Для каждого топика платформа поддерживает партиционированный журнал, схематично представленный на Рис.1.1.

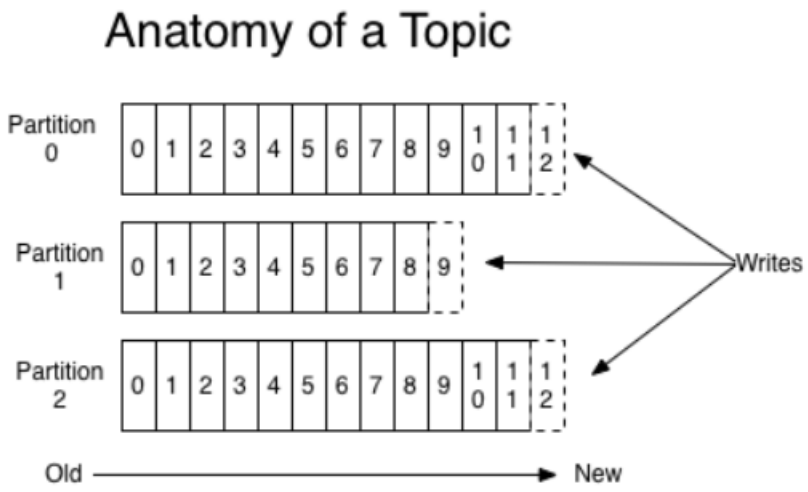


Рис.1.1.: Партиционированный журнал

Каждая партиция представляет собой упорядоченную неизменяемую последовательность записей, которая постоянно добавляется в структурированный журнал. Каждой записи в партиции присваивается порядковый номер *id*, называемый смещением (*offset*), который однозначно идентифицирует каждую запись.

Платформа **ADS** надежно сохраняет все опубликованные записи в соответствии с настройкой периода их хранения. Например, если политика хранения установлена на два дня, то в течение двух дней после публикации запись доступна для потребления, после чего она удаляется с целью освобождения места. Производительность **ADS** фактически постоянна по отношению к размеру данных, что обеспечивает возможность хранения знаписей в течение длительного времени (Рис.1.2.).

Фактически, метаданные, сохраненные для каждого потребителя, являются его смещением или положением в журнале. Это смещение контролируется самим потребителем: обычно потребитель линейно продвигает свое смещение при считывании записи, но так как его позиция контролируется им самим, то он

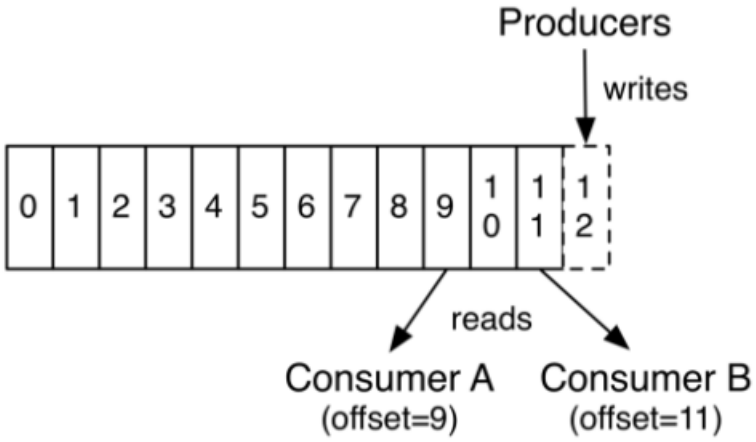


Рис.1.2.: Логика смещения

может считывать записи в любом порядке. Например, потребитель может вернуться к более старому смещению для повторной обработки данных или перейти к самой последней актуальной записи и начать считывание с настоящего момента.

Такое сочетание функций означает, что потребители **ADS** могут приходить и уходить без особого влияния на кластер и на других потребителей. Например, можно использовать инструменты командной строки для считывания данных с конца любого топика без какого-либо влияния на то, что считывается другими потребителями.

Партиции в журнале служат нескольким целям. Во-первых, они позволяют журналу масштабироваться сверх размера, который помещается на одном сервере. Каждая отдельная партиция располагается на конкретном сервере, но топик может иметь много партиций и располагаться на нескольких серверах для возможности обработки произвольного количества данных. Во-вторых, партиции действуют как единица параллелизма.

Партиции журнала распределяются по серверам кластера **ADS**, при этом каждый сервер обрабатывает данные и запросы к определенным партициям. Каждая партиция реплицируется на настраиваемое число серверов для обеспечения отказоустойчивости.

У каждой партиции всегда имеется один сервер, выступающий в качетсве “лидера”. Лидер обрабатывает все запросы на чтение и запись для партиции, а остальные сервера пассивно реплицируют изменения лидера. Если лидер выходит из строя, один из брокеров автоматически становится новым лидером.

ADS MirrorMaker обеспечивает поддержку георепликации для кластеров. С помощью **MirrorMaker** сообщения реплицируются через несколько центров обработки данных или облачных сервисов, что можно использовать в активных/пассивных сценариях резервного копирования и восстановления или в активных/активных сценариях для размещения данных ближе к пользователям, а так же с целью поддержки требований к местоположению данных.

Поставщики публикуют данные по топикам по своему усмотрению и отвечают за выбор того, какую запись назначить для какой партиции. Это может быть сделано в циклическом режиме для балансировки нагрузки, или это может быть сделано в соответствии с какой-либо семантической функцией разбиения (например, на основе некоторого ключа в записи).

Потребители относят себя к группе потребителей, и каждая запись, опубликованная в топике, доставляется каждому инстансу потребителя, группа которого подписана на данный топик. При этом инстансы потребителя могут находиться на отдельных процессах или машинах. Если все инстансы потребителя имеют одну и ту же группу, то записи эффективно балансируются. А в случае если инстансы потребителя имеют разные группы, то каждая запись передается во все потребительские процессы (Рис.1.3.).

На рисунке приведен пример двухсерверного кластера **ADS** с четырьмя партициями (*P0-P3*) и с двумя

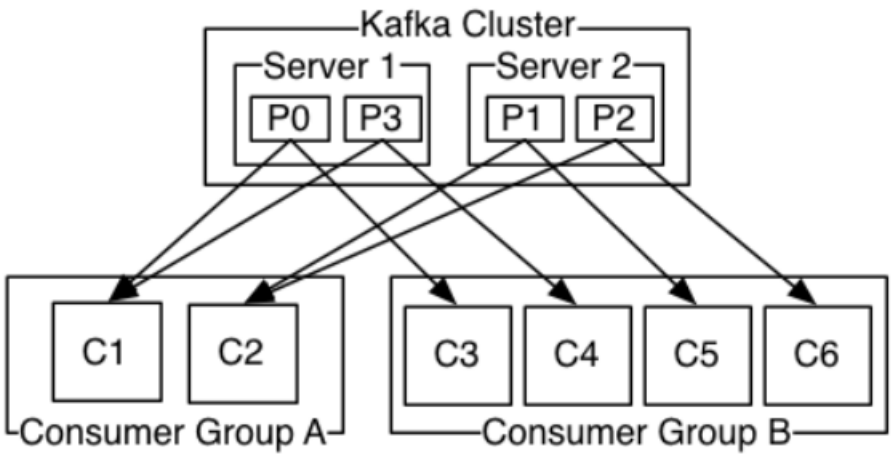


Рис.1.3.: Группы потребителей

группами потребителей. Группа потребителей A имеет два экземпляра потребителей, группа B – четыре.

Чаще всего топики имеют небольшое количество групп потребителей – по одной для каждого “логического подписчика” (“logical subscriber”). Каждая группа состоит из множества инстансов потребителей для обеспечения масштабируемости и отказоустойчивости. Это не что иное, как семантика “издатель-подписчик” (“publish-subscribe”), где подписчик представляет собой не один процесс, а группу потребителей.

Реализация способа считывания в **ADS** заключается в разделении записей журнала на партиции, исходя из экземпляров потребителя, чтобы каждый экземпляр был исключительным потребителем “изрядной доли” (“fair share”) партиций в любой момент времени. Процесс поддержания членства в группе динамически обрабатывается протоколом **ADS**. Если к группе присоединяются новые экземпляры, они принимают некоторые партиции от других членов группы; если экземпляр удаляется, его партиции распределяются по остальным экземплярам.

ADS предоставляет только общий порядок записей внутри партиции, а не между партициями в топике. Упорядочивание по разделам в сочетании с возможностью разбиения данных по ключам для большинства приложений является достаточным. Однако если требуется полный порядок по записям, это может быть достигнуто с помощью топика, имеющего только одну партицию, хотя это будет означать только один потребительский процесс для каждой группы потребителей.

ADS можно развернуть как *multi-tenant* решение. Этот режим настраивает, в какой топик могут записываться данные, а из какого считываться. Существует также операционная поддержка квот. Администраторы могут определять и применять квоты на запросы для управления ресурсами брокера, которые используются клиентами.

Глава 2

Гарантии

ADS обеспечивает следующие гарантии:

- Сообщения, отправленные поставщиком в определенную партицию топика, добавляются в таком же порядке. То есть, если записи *M1* и *M2* отправляются одним поставщиком, и сначала отправляется *M1*, тогда *M1* имеет меньшее смещение, чем *M2*, и появляется в журнале раньше;
- Инстанс потребителя видит записи в том же порядке, в котором они хранятся в журнале;
- Для топика с коэффициентом репликации *N* допустимо до *N-1* сбоев сервера без потери записей, зафиксированных в журнале.

Глава 3

Рекомендации по использованию

3.1 ADS как Messaging System

Как понятие о потоках **ADS** сравнивается с традиционной корпоративной системой обмена сообщениями?

Обмен сообщениями традиционно имеет две модели: “очередь” и “публикация-подписка”. В очереди группа потребителей может читать с сервера, и каждая запись переходит к одному из них; в модели “публикация-подписка” запись передается всем потребителям.

Каждая из этих двух моделей имеет свои плюсы и минусы. Преимущество очереди заключается в том, что она допускает разделение обработки данных на несколько инстансов потребителей, что позволяет масштабировать обработку. Но при этом очереди не являются многопотребительскими – как только один процесс считывает данные, они удаляются. Модель обмена сообщениями “публикация-подписка” позволяет передавать данные нескольким процессам, но при этом не имеет возможности масштабирования, так как каждое сообщение отправляется каждому подписчику.

Концепция группы потребителей в **ADS** обобщает эти две модели. Как и в случае с очередью, группа потребителей позволяет разделять обработку по совокупности процессов (по членам группы потребителей). А как в случае с моделью “публикация-подписка”, **ADS** позволяет передавать сообщения нескольким группам потребителей.

Преимущество модели **ADS** заключается в том, что каждый топик имеет оба этих свойства – он может масштабировать обработку, а также является многопотребитительским – необходимость выбора той или иной модели отпадает.

К тому же **ADS** имеет более мощные гарантии упорядочения, чем традиционная система обмена сообщениями.

Традиционная очередь сохраняет записи на сервере по порядку, и передача записей с сервера осуществляется в порядке их хранения на нем. Однако если считывание из очереди производится несколькими потребителями, то, несмотря на то, что сервер ведет записи по порядку, данные доставляются потребителям асинхронно. Фактически это означает, что при параллельном считывании упорядоченность записей теряется. Системы обмена сообщениями часто обходят эту проблему, используя понятие “исключительный потребитель” (“exclusive consumer”), которое позволяет только одному процессу считывать из очереди, но это говорит о том, что параллелизм в обработке отсутствует.

В платформе **ADS** эта проблема решена. Имея понятие “параллелизм – партиция – в рамках топика” (“parallelism – the partition – within the topics”), **ADS** может обеспечить как гарантии упорядоченности, так и балансировку нагрузки над группами потребительских процессов. Это достигается путем назначения партиций в топике для потребителей соответствующей группы, чтобы каждая партиция считывалась ровно одним потребителем в группе. При этом гарантируется, что потребитель является единственным читателем этой

партиции и потребляет данные по порядку. Поскольку существует много партиций, нагрузка балансируется по всем экземплярам потребителя. Также следует обратить внимание, что в группе не может быть экземпляров потребителей больше, чем партиций.

3.2 ADS как Storage System

Любая очередь, которая позволяет публиковать сообщения, не связанные с их потреблением, эффективно может быть использована как система хранения на лету поступающих сообщений. Отличие платформы **ADS** заключается в том, что она является не только брокером сообщений, но и системой хранения данных.

Помещаемые в **ADS** данные записываются на диск и реплицируются для обеспечения отказоустойчивости. **ADS** позволяет поставщикам дождаться подтверждения операции, так что запись не считается полной, пока она не будет полностью реплицирована и гарантированно сохранена, даже если сервер вышел из строя.

Дисковые структуры **ADS** хорошо масштабируются: платформа функционирует одинаково, не зависимо от объема постоянных данных на сервере – будь то *50 КБ* или *50 ТБ*.

С целью использования **ADS** в качестве хранилища и предоставления клиентам возможности контролирования позиции чтения, платформу можно рассматривать как своего рода распределенную файловую систему специального назначения, предназначенную для высокопроизводительного хранения данных с низкой задержкой коммитов в журнал, а также для их репликации и распространения.

3.3 ADS – Stream Processing

Недостаточно просто читать, записывать и хранить потоки данных, важно обеспечить обработку потоков в реальном времени.

В **ADS** потоковый процессор – это все, что принимает непрерывные потоки данных из входных топиков, выполняет некоторую обработку и создает непрерывные потоки данных в выходные топики.

Например, приложение розничной торговли может принимать входные потоки по продажам и отгрузкам и выводить поток с корректировками цен, вычисленными на основе входных данных.

Простую обработку можно выполнять напрямую, используя Consumer API и Producer API. Однако для более сложных преобразований **ADS** предоставляет Streams API. Он позволяет создавать приложения с нетривиальной обработкой, выполняющей агрегацию потоков или объединяющей их вместе. Это помогает решить сложные проблемы, с которыми сталкивается подобный тип приложений: обработка неупорядоченных данных, переработка входных данных при изменении кода, выполнение вычислений с учетом состояния и т.д.

Streams API построен на основных примитивах **ADS**: он использует Producer API для ввода, использует **Kafka** для хранения состояний и применяет механизм группы для обеспечения отказоустойчивости среди инстансов потокового процессора.

Глава 4

Концепция потоков NiFi

Ключевые концепции **NiFi** тесно связаны с основными идеями потокового программирования (Flow Based Programming – FBP). Далее в таблице приведены некоторые из основных концепций **NiFi** и их сопоставление с **FBP**.

Таблица4.1.: Сопоставление концепций NiFi и FBP

NiFi Term	FBP Term	Описание
FlowFile	Information Packet	FlowFile представляет каждый объект, перемещающийся через систему, и для каждого из них NiFi отслеживает список атрибутов (пары ключ/значение) и связанного с ним содержимое с количеством байт от нуля и больше
FlowFile Processor	Black Box	Процессоры, выполняющие задание. В терминах eip процессор выполняет некоторую комбинацию маршрутизации, преобразования и обмена информацией между системами. Процессоры имеют доступ к атрибутам предоставленного FlowFile и его контенту. Процессоры могут работать с ноль и более FlowFiles в данном блоке задания, либо выполнять работу (commit), либо отменять результат ее выполнения откатом (rollback)
Connection	Bounded Buffer	Соединения выступают в роли фактической связи между процессорами. Они действуют как очереди и позволяют разным процессам взаимодействовать с разной скоростью. Очереди могут быть распределены по приоритетам и иметь верхние границы нагрузки, инициализирующие процесс *back pressure*
Flow Controller	Scheduler	Контроллер потока хранит знание о том, как соединены процессы, управляет и распределяет потоки, которые используются процессами. Flow Controller выступает в качестве брокера, облегчающего обмен FlowFiles между процессорами
Process Group	subnet	Группа процессов представляет собой определенный набор процессоров и их соединений, которые могут принимать данные через порты для входящих соединений и отправлять их через выходные порты. Таким образом, группы процессов позволяют создавать совершенно новые компоненты просто посредством композиции других компонентов

Такая модель проектирования (схожая с seda) предоставляет множество преимуществ по созданию мощных и масштабируемых потоков данных, например:

- Хорошо подходит для визуального создания и управления направленными графами процессоров;
- По своей сути является асинхронной, что обеспечивает очень высокую пропускную способность и естественную буферизацию даже при колебаниях скорости обработки и потока;
- Обеспечивает высококонкурентную модель без необходимости участия разработчика в вопросах потокобезопасности;
- Способствует развитию связанных и слабосвязанных компонентов, которые в последствии могут быть повторно использованы в других контекстах и способствовать тестированию;
- Соединения с верхними и нижними границами по ресурсам делают такие критические функции, как *back pressure* и *pressure release*, закономерными и интуитивно понятными;
- Обработка ошибок становится ожидаемым результатом успешного выполения конкретного сценария процесса (happy-path), а не catch-all;
- Точки входа и выхода данных, а также перемещение потоков в системе, понятны и легко отслеживаются.