СБЕР УНИВЕРСИТЕТ

Школа Java Middle Developer

Kafka

Потоковая обработка

Содержание

1. Основы (паттерны) потоковой обработки данных;
2. Kafka Streams.

Что такое потоковая обработка?

Потоки данных

- ✓Упорядоченность потоков событий;
- ✓Неизменяемость записей данных;
- ✓Повторяемость потоков событий.

Потоковая обработка

- Запрос/ответ
- Пакетная обработка
- Потоковая обработка

Основные понятия потоковой обработки

Время

- Время события
- Время добавления информации в журнал
- Время обработки

Не забывайте о часовых поясах!

ZoneDateTime | OffsetDateTime | LocalDateTime

Состояние

- Локальное (внутреннее) состояние
- Внешнее состояние

Таблично-потоковый дуализм

Поток изменения наличия товара

Поставка	Синие туфли	300
	Красные туфли	300
	Зеленые туфли	300
Продажа	Синие туфли	299
	Красные туфли	299
Возврат	Синие туфли	300
	Зеленые туфли	299

Таблица/материализованное представление текущего состояния склада

Красные туфли	299
Синие туфли	300
Зеленые туфли	299

Временные окна

Важно понимать, какой именно тип временного окна нам требуется.

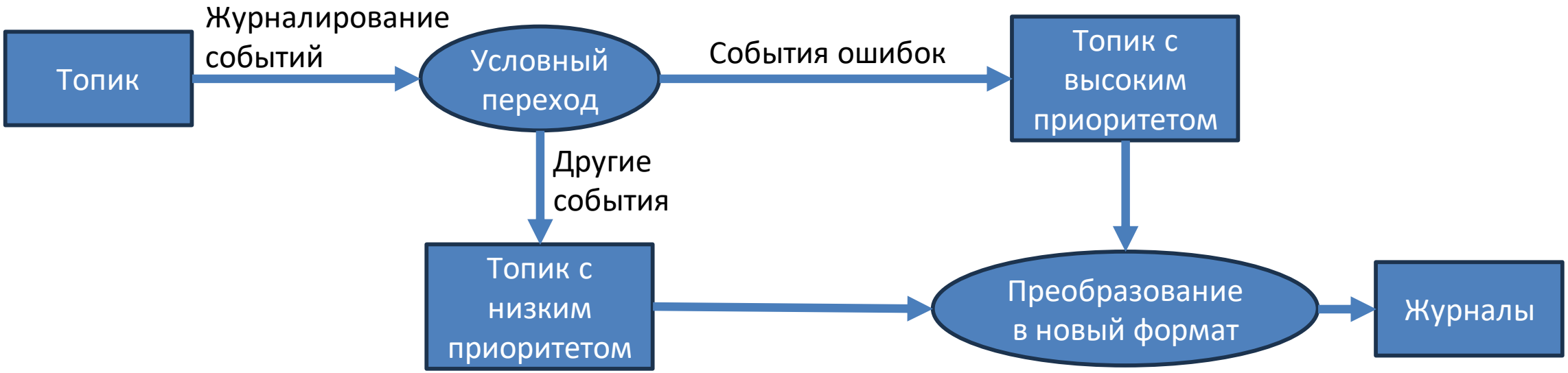
Например, при вычислении скользящих средних необходимо знать следующее:

- Размер окна
- Насколько часто окно сдвигается
- В течение какого времени сохраняется возможность обновления окна

Паттерны проектирования потоковой обработки

Обработка событий по отдельности

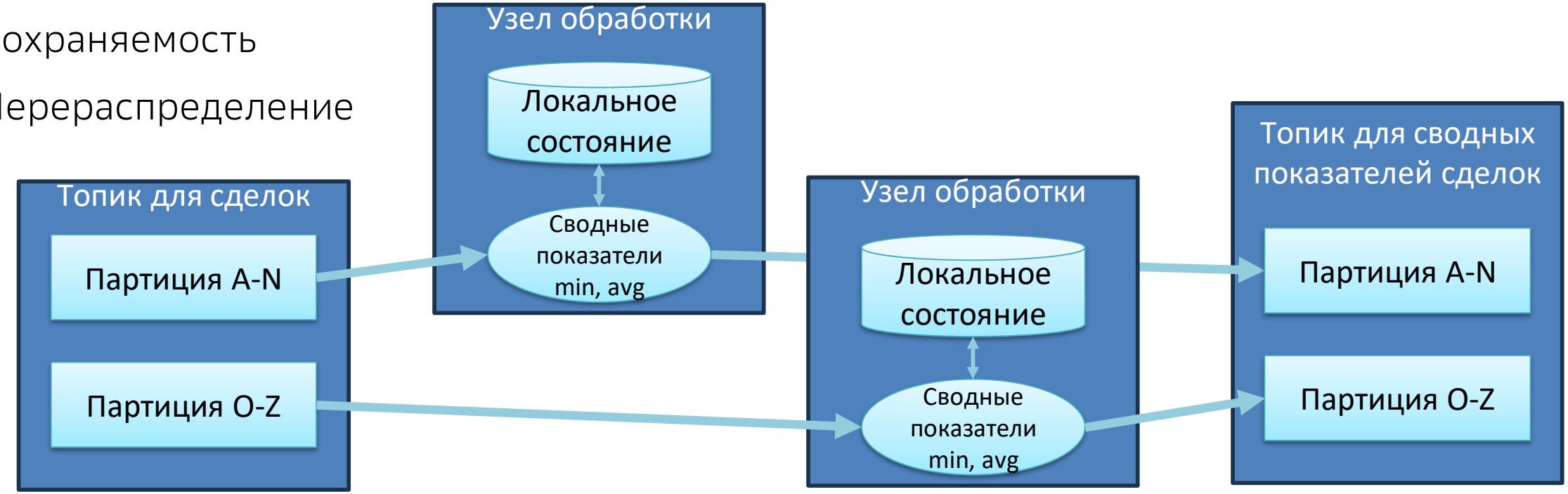
- ✓ Простейший паттерн потоковой обработки - обработка каждого события по отдельности
- ✓ Приложения, построенные по этому паттерну, могут не хранить внутри себя состояние, поскольку события могут обрабатываться по отдельности
- ✓ После сбоя восстанавливать состояние не нужно, можно просто делегировать обработку событий другому экземпляру приложения
- ✓ Для этого паттерна вполне достаточно простого производителя и потребителя



Обработка с использованием локального состояния

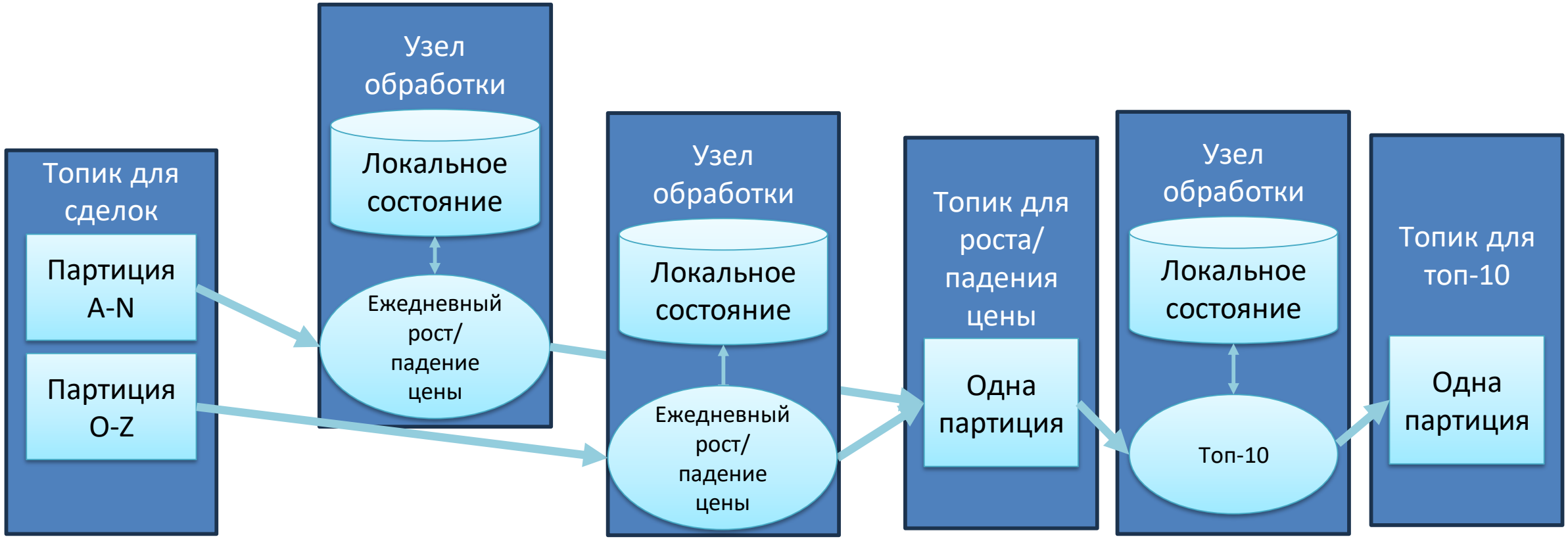
Приложения потоковой обработки существенно усложняются при наличии локального состояния, так как возникает несколько проблем, которые должно решить такое приложение:

- Использование памяти
- Сохраняемость
- Перераспределение



Многоэтапная обработка/повторное разделение на партиции

- ✓ Несколько этапов
- ✓ На первом этапе обработка локальных событий
- ✓ На последующих объединение и повторная обработка результата

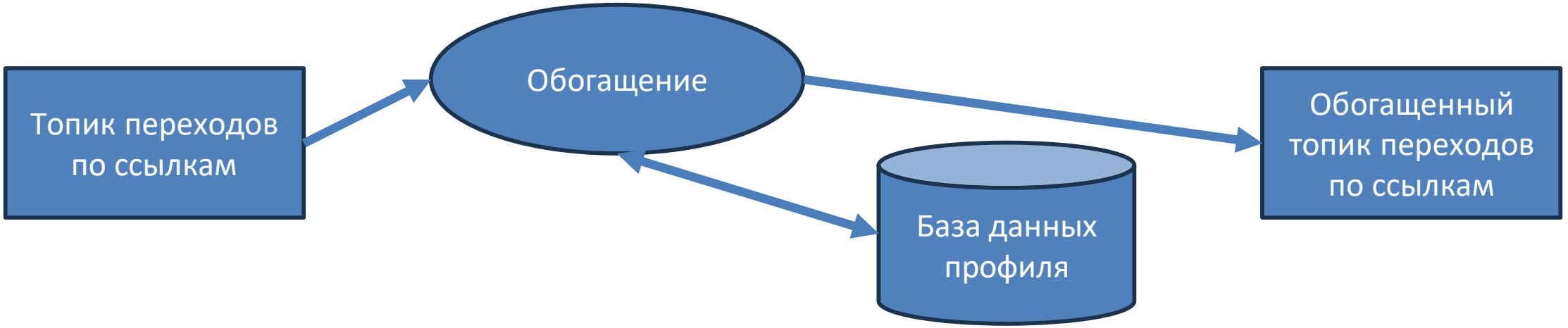


Обработка с применением внешнего справочника: соединение потока данных с таблицей

Проблема: внешний справочник существенно увеличивает время обработки каждой записи

Решение: т.к. речь о справочнике, то можно применять кэш, но управлять кэшем тоже может быть непросто – как предотвратить устаревание данных в нём?

Реализовать CDC (change data capture) для справочника.



Объединение потоков

- Объединение двух потоков данных означает объединение полной истории событий и поиск соответствий событий из одного потока событиям из другого, относящимся к тем же временным окнам и с такими же ключами
- Объединение потоков называют также оконным объединением (windowed-join)

Внеочередные события

Потоковые приложения должны корректно обрабатывать несвоевременно поступившие события:

- Распознать несвоевременное поступление события
- Определиться с интервалом времени, в течение которого оно будет пытаться синхронизировать внеочередные события
- Обладать достаточными возможностями для синхронизации данного события
- Иметь возможность обновить результаты

Повторная обработка

Существует два варианта реализации паттерна повторная обработка событий:

- У нас появилась новая версия приложения потоковой обработки, и нужно организовать обработку этой версией того же потока событий, который обрабатывает старая, получить новый поток результатов, не замещающий первой версии, сравнить две версии результатов и в какой-то момент перевести клиентов на использование новых результатов вместо существующих
- В существующее приложение потоковой обработки вкралась программная ошибка. Мы ее исправили и хотели бы заново обработать поток событий и вычислить новые результаты

Повторная обработка

Для работы двух версий приложения потоковой обработки, записывающих два потока результатов, достаточно выполнить следующее:

- Развернуть новую версию приложения в качестве новой группы потребителей
- Настроить новую версию так, чтобы она начала обработку с первого смещения исходных топиков
- Продолжить работу нового приложения и переключить клиентские приложения на новый поток результатов после того, как новая версия выполняющего обработку задания наверстает отставание

Kafka Streams в примерах

В Apache Kafka есть два потоковых API:

- Низкоуровневый Processor API - позволяет создавать собственные преобразования
- Высокоуровневый Streams DSL - позволяет задавать приложение потоковой обработки путем описания последовательности преобразований событий потока

Подсчет количества слов

```
<dependency>
   <groupId>org.apache.kafka</groupId>
   <artifactId>kafka-streams</artifactId>
   <version>3.7.1</version>
</dependency>

<dependency>
   <groupId>org.apache.kafka</groupId>
   <artifactId>kafka-clients</artifactId>
   <version>3.7.1</version>
</dependency>
```

Подсчет количества слов

```
Properties props = new Properties();
props.put(StreamsConfig.APPLICATION_ID_CONFIG, "wordcount"); // 1
props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost: 9092"); // 2
props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName()); // 3
props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
```

Подсчет количества слов

```
StreamsBuilder builder = new StreamsBuilder(); // 1
KStream<String, String> source = builder.stream("wordcount-input");

final Pattern pattern = Pattern.compile("\\W+");
KStream counts = source.flatMapValues(value -> Arrays.asList(pattern.split(value.toLowerCase()))) // 2
      .map((key, value) -> new KeyValue<Object, Object>(value, value))
      .filter((key, value) -> (!value.equals("the"))) // 3
      .groupByKey() // 4
      .count(Named.as("CountStore"))
      .mapValues(value -> Long.toString(value))
      .toStream(); // 5

counts.to("wordcount-output"); // 6
```

Подсчет количества слов

```
Topology topology = builder.build();
KafkaStreams streams = new KafkaStreams(topology, props); // 1
streams.start(); // 2

Thread.sleep(5000L);
streams.close(); // 3
```

Сводные показатели фондовой биржи надежности системы

Следующий пример сложнее — мы прочитаем поток событий биржевых операций, включающий символы акций, цену и величину предложения. В биржевых операциях цена предложения (ask price) — это то, сколько просит за акции продавец, а цена заявки (bid price) - то, что готов заплатить покупатель. Величина предложения (ask size) - число акций, которое продавец согласен продать по данной цене.

Также создадим выходные потоки данных, содержащие несколько оконных сводных показателей:

- наилучшую, то есть минимальную цену предложения для каждого пятисекундного окна;
- число сделок для каждого пятисекундного окна;
- среднюю цену предложения для каждого пятисекундного окна.

Сводные показатели фондовой биржи надежности системы

```
Properties props = new Properties();
props.put(StreamsConfig.APPLICATION_ID_CONFIG, "stockstat");
props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,
Serdes.String().getClass().getName());
props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
TradeSerde.class.getName());
```

```
static public final class TradeSerde extends WrapperSerde<Trade> {
    public TradeSerde() {
        super(new JsonSerializer<Trade>(), new JsonDeserializer<Trade>(Trade.class));
    }
}
```

Сводные показатели фондовой биржи

```
StreamsBuilder builder = new StreamsBuilder();
KStream<String, String> source = builder.stream("source");

KStream<TickerWindow, TradeStats> stats = source.groupByKey() // 1
      .aggregate(TradeStats::new, // 2
            (key, value, tradestats) -> tradestats.add(value), // 3
            Materialized.with(Serdes.String(), Serdes.serdeFrom(new Serializer<TradeStats>()) // 6
      .toStream((key, value) -> new TickerWindow(key.key(), key.window().start())) // 7
      .mapValues((trade) -> trade.computeAvgPrice()); // 8

stats.to("stockstats-output”, Produced.with(Serdes.String(), Serdes.serdeFrom(new Serializer<TradeStats>()); // 9
```

Обогащение потока событий перехода по ссылкам

Сгенерируем поток имитационных щелчков по ссылкам, поток обновлений таблицы базы данных с фиктивными профилями, а также поток операций поиска в Сети. Затем соединим все три потока, чтобы получить полный обзор деятельности всех пользователей.

Конфигурация приложения такая же, как в предыдущих примерах.

Обогащение потока событий перехода по ссылкам

```
StreamsBuilder builder = new StreamsBuilder();
KStream<String, String> source = builder.stream("source");
KStream<Integer, PageView> views = builder.stream(Serdes.Integer(),
      new PageViewSerde(),
      Constants.PAGE_VIEW_TOPIC);
KStream<Integer, Search> searches = builder.stream(Serdes.Integer(),
      new SearchSerde(),
      Constants.SEARCH_TOPIC);
KTable<Integer, UserProfile> profiles = builder.table(Serdes.Integer(),
      new ProfileSerde(),
      Constants.USER_PROFILE_TOPIC,
      "profile_store");
KStream<Integer, UserActivity> viewsWithProfile = views.leftJoin(profiles,
      (page, profile) -> new UserActivity(profile.getUserID(), profile.getUserName(), profile.getZipcode(), profile.getInterests(), "", page.getPage()));

KStream<Integer, UserActivity> userActivityKStream =
      viewsWithProfile.leftJoin(searches, (userActivity, search) ->
                  userActivity.updateSearch(search.getSearchTerms()),
            JoinWindows.of(1000),
            Serdes.Integer(),
            new UserActivitySerde(),
            new SearchSerde());
KafkaStreams streams = new KafkaStreams(builder.build(), props);
streams.start();
```

Kafka Streams: обзор архитектуры

Построение топологии

Масштабирование топологии

- Можно запустить приложение Kafka Streams на одной машине в многопоточном режиме или на нескольких машинах — в любом случае обработкой данных будут заниматься все активные потоки выполнения приложения
- Движок Streams распараллеливает выполнение топологии, разбивая ее на задачи
- Каждая задача отвечает за какое-то подмножество партиций
- Для каждого прочитанного события задача выполняет по порядку все подходящие для этой партиции шаги обработки, после чего записывает результаты в приемник
- Задачи - базовая единица параллелизма в Kafka Streams

Зависимости между задачами

- При соединении двух потоков данных для получения результата понадобятся данные из партиции каждого из потоков
- Фреймворк Kafka Streams решает эту проблему за счет назначения всех необходимых для одного соединения партиций одной задаче, так что задачи могут читать данные из всех нужных партиций и выполнять соединение независимо друг от друга
- Еще один пример возникновения зависимостей между задачами - случай, когда для приложения требуется повторное секционирование
- Kafka Streams выполняет повторное разделение путем записи событий в новый топик с новыми ключами и партициями

Как пережить отказ

- Фреймворк Kafka Streams использует предоставляемую Kafka координацию потребителей с целью обеспечения высокой доступности для задач
- Если задача завершилась неудачей, но есть другие активные потоки или экземпляры потокового приложения, ее можно перезапустить в одном из доступных потоков

Сценарии использования потоковой обработки

- Обслуживание клиентов
- Интернет вещей
- Обнаружение мошенничества

Как выбрать фреймворк потоковой обработки

- Система ввода и обработки данных
- Система, реагирующая в течение нескольких миллисекунд
- Асинхронные микросервисы
- Анализ данных в режиме псевдореального времени

Как выбрать фреймворк потоковой обработки

- Если вы пытаетесь решить задачу ввода и обработки данных, лучше подумать еще раз, нужна ли вам система потоковой обработки или подойдет более примитивная система, ориентированная именно на ввод и обработку данных, например Kafka Connect
- Если вы пытаетесь решить задачу, требующую реакции в течение нескольких миллисекунд, лучше также еще раз подумать, прежде чем выбрать систему потоковой обработки
- При создании асинхронных микросервисов вам понадобится система потоковой обработки
- При создании сложной системы для аналитики вам также понадобится система потоковой обработки с хорошей поддержкой локального хранилища

Как выбрать фреймворк потоковой обработки

- Удобство эксплуатации системы
- Простота использования API и легкость отладки
- Облегчение жизни
- Помощь сообщества разработчиков

Спасибо за внимание