

avito.tech

Предтеченская Валентина

Apache Flink

Distributed, Stateful, Realtime

устройство, грабли, применимость



Apache Flink Documentation

Apache Flink is a framework and distributed processing engine for stateful computations over *unbounded* and *bounded* data streams.

Distributed

processing engine

как распределяются
данные между
машинами и
процессами ?

Stateful

computations

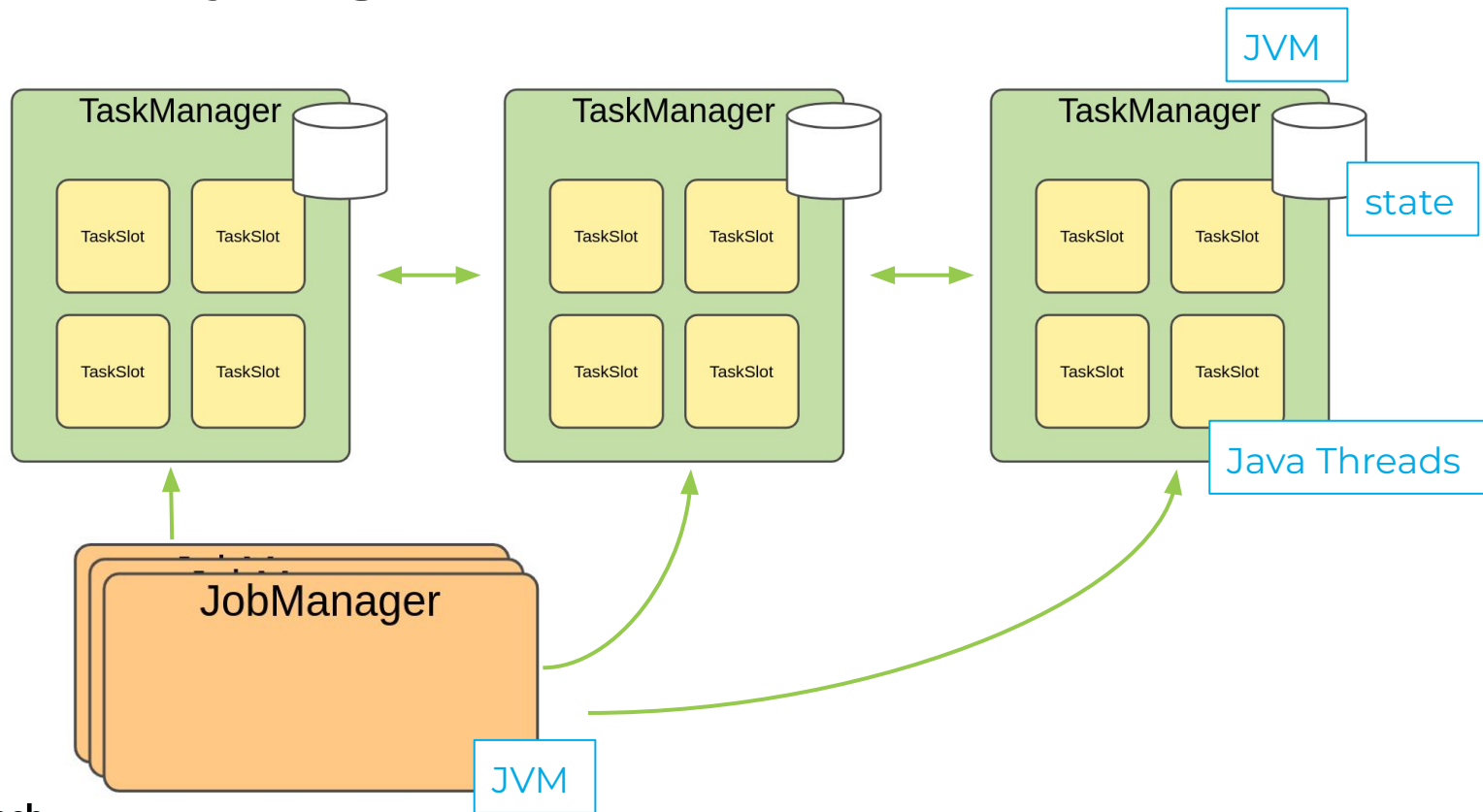
где живет стейт и как
правильно с ним
работать ?

Realtime

unbounded
data stream

обработка
бесконечного потока
событий происходит в
режиме реального
времени: что такое
время и как с ним
правильно работать ?

FLINK КЛАСТЕР



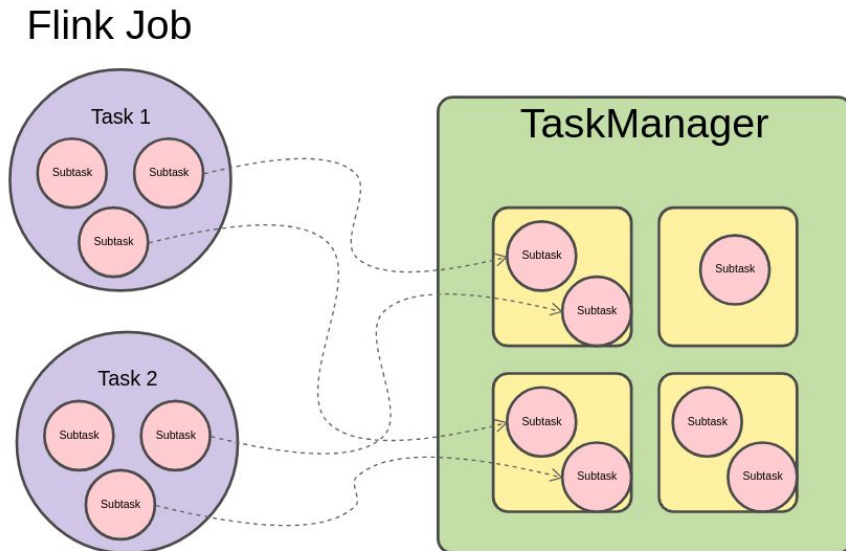
FLINK КЛАСТЕР

- ▶ **Task Manager** состоит из Task Slots
- ▶ **Task Slot** - условная единица ресурса

- ▶ **Job** состоит из Tasks
- ▶ **Task** - условная единица работы - состоит из (параллельных) Subtasks

- ▶ **Subtask** выполняется в Task Slot в одном потоке (Thread)

- ▶ В одном Task Slot могут выполняться Subtasks разных Task-ов



PART I


DISTRIBUTED



СОБЫТИЯ ПОЛЬЗОВАТЕЛЯ



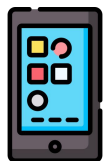
Сено в тюках
120 ₽
Сена люцерны в тюках. Понос 2023. Возможно доставка! Доставка от 100 штук!
Товары для животных
Краснодарский край, Белореченский р-н, Белореченское городское поселение, Белореченск
Несколько секунд назад



Гуси домашние
1 500 ₽
Продам двух домашних гусей на племя. Гусыни хорошо неслись и сами выводили гусят. Живут в хозяйстве, но их жалко рубить.
Другие животные
Воронежская область, Новоаннинский р-н, Рождественско-Хавское сельское поселение, Рождественск-Хавский район
Несколько секунд назад



Зааненская коза
4 700 ₽
Коза котная через месяц+- родит. Козе примерно 1,5 года. Один рог сломан. Торг уместен.
Другие животные
Ростовская область, Азовский р-н, Кружляное сельское поселение, с. Стефандинодар
Несколько секунд назад



kafka



```
{  
  "id": 1,  
  "uid": 30,  
  "ip": "87.90.89.74",  
  "itemId": 955,  
  "timestamp": 1691482581510  
}
```

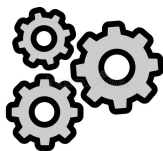
ID пользователя

ЗАДАЧА: ПОДСЧЕТ КЛИКОВ

```
{  
  "id": 1,  
  "uid": 30,  
  "ip": "87.90.89.74",  
  "itemId": 955,  
  "timestamp": 1691482581510  
}
```

key - ID пользователя
value - количество
кликов

```
"key": 30,  
"value": 10
```



```
@JsonIgnoreProperties(ignoreUnknown = true)
public class Event {

    @JsonProperty("id")
    private Integer eventId;
    private Integer uid;
    private String ip;
    private Integer itemId;
    private Long timestamp;

    public Event() {
    }

    public Event(Integer eventId, Integer uid, String ip, Integer itemId, Long timestamp) {
        this.eventId = eventId;
        this.uid = uid;
        this.ip = ip;
        this.itemId = itemId;
        this.timestamp = timestamp;
    }
    // ... getters && setters
}
```




```

public class ExampleJob1 {

    public static void main(String[] args) throws Exception {

        KafkaSource<Event> eventsSource = KafkaSource.<Event>.builder()
            .setBootstrapServer("localhost:9092")
            .setTopic("example-smartdata")
            .setDeserializer(new EventDeserializer())
            .setStartingOffsets(OffsetsInitializer.earliest())
            .build();

        KafkaSink<Tuple2<Integer, Integer>> eventsSink = KafkaSink.<Tuple2<Integer, Integer>>.builder()
            .setBootstrapServer("localhost:9092")
            .setRecordSerializer(new ResultSerializer("example-smartdata-result"))
            .build();

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

        env.fromSource(eventsSource, WatermarkStrategies.noWatermarks(), "events")
            .name("Source events")
            .uid("source-events")

            .keyBy(Event::getUid)
            .process(new CounterFunction())
            .name("Count events")
            .uid("count-events")

            .sinkTo(eventsSink)
            .name("Sink result")
            .uid("sink-result");

        env.execute("example-job");
    }
}

```



```

public class CounterFunction extends KeyedProcessFunction<Integer, Event, Tuple2<Integer, Integer>> {

    private ValueState<Integer> counterState;

    @Override
    public void open(Configuration parameters) {
        ValueStateDescriptor<Integer> counterStateDesc = new ValueStateDescriptor<>("counter",
Integer.class);
        counterState = getRuntimeContext().getState(counterStateDesc);
    }

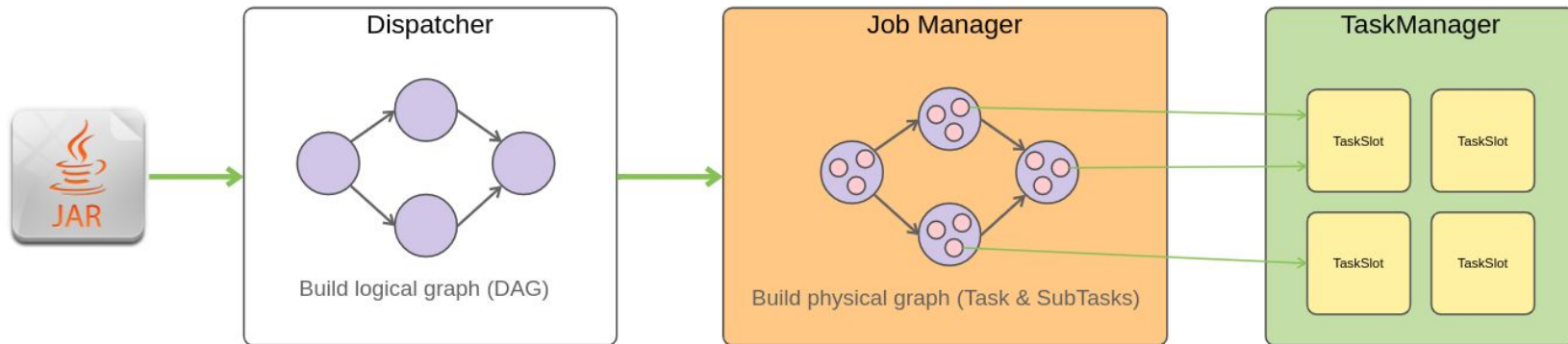
    @Override
    public void processElement(Event value, Context ctx, Collector<Tuple2<Integer, Integer>> out throws
Exception {
        Integer counterValue = counterState.value();
        if (counterValue == null) {
            counterValue = 0;
        }
        counterState.update(++counterValue);
        out.collect(new Tuple2<>(ctx.getCurrentKey(), counterValue));
    }
}

```



ЧТО ПРОИСХОДИТ ПРИ ДЕПЛОЕ ?

- ▶ **Dispatcher** получает JAR файл и строит логический **Job Graph**
- ▶ Построенный граф передается **Job Manager** вместе с кодом и конфигурацией (parallelism)
- ▶ **Job Manager** строит физический граф в соответствии с конфигурацией и параллелизмом
- ▶ Запрашивается необходимое количество **Task Slots**
- ▶ SubTasks передаются на **Task Managers** в соответствующие **Task Slots**



JOB GRAPH

```
env.fromSource(eventsSource, WatermarkStrategy.noWatermarks(), "events")
    .name("Source events")
    .uid("source-events")

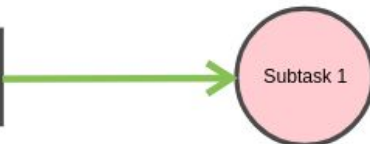
    .keyBy(Event::getUid)
    .process(new CounterFunction())
    .name("Count events")
    .uid("count-events")

    .sinkTo(eventsSink)
    .name("Sink result")
    .uid("sink-result");
```

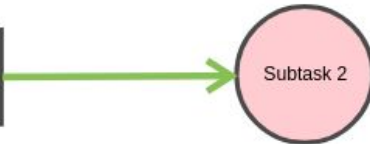


PARALLELISM < KAFKA PARTITIONS

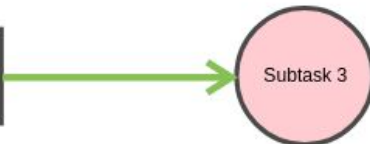
partition 1



partition 2



partition 3



7 subtasks (kafka consumers) ничего не делают

JOB GRAPH

```
env.fromSource(eventsSource, WatermarkStrategy.noWatermarks(), "events")
    .name("Source events")
    .uid("source-events")
    .setParallelism(3)

    .keyBy(Event::getUid)
    .process(new CounterFunction())
    .name("Count events")
    .uid("count-events")

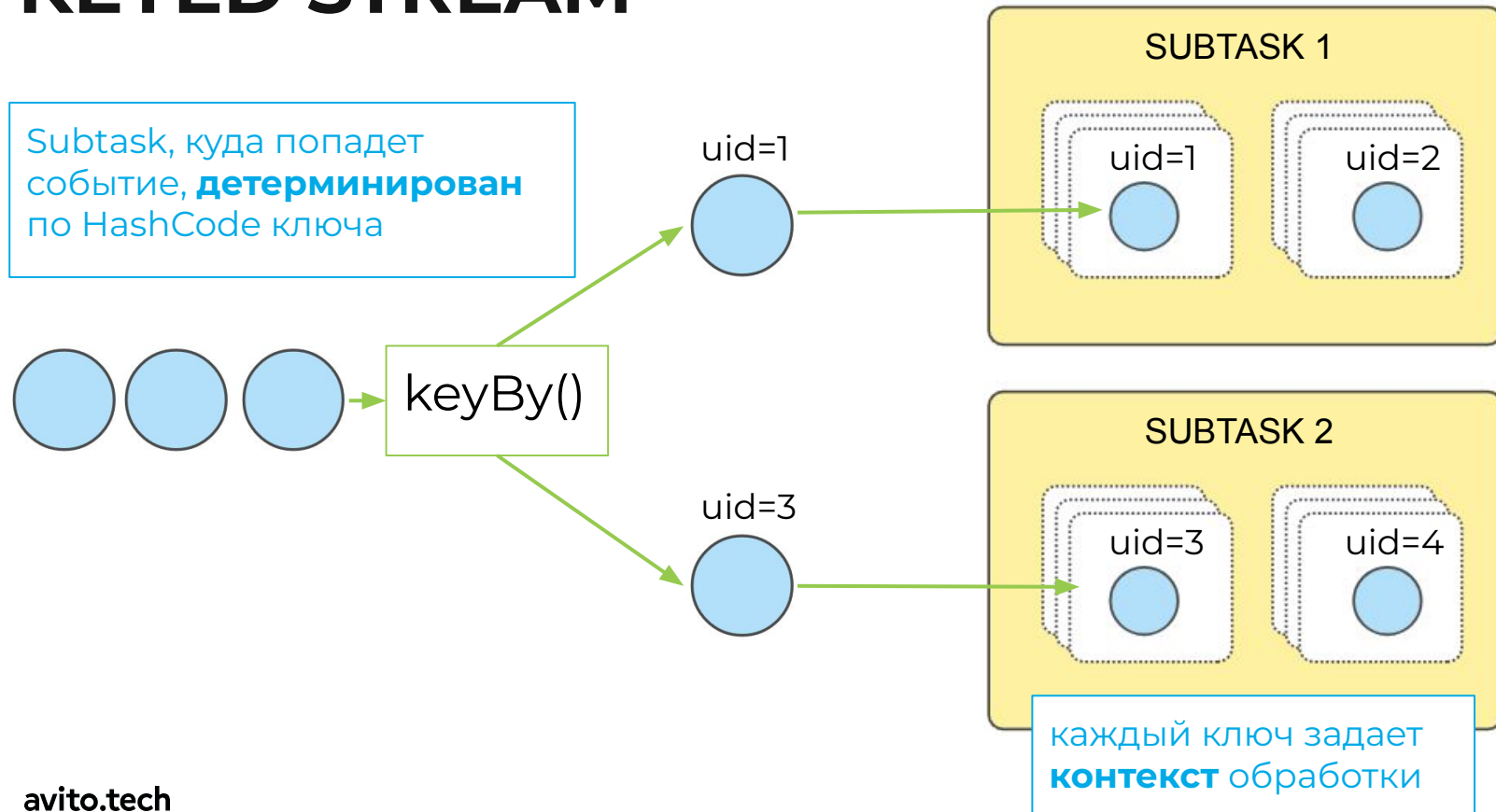
    .sinkTo(eventsSink)
    .name("Sink result")
    .uid("sink-result");
```



КАК ВЫБРАТЬ ПАРАЛЛЕЛИЗМ ?

- 01 эмпирически (нужно нагрузочное тестирование) выбираем общий параллелизм
- 02 закладываем немного сверху (*HEADROOM*) на случай увеличения трафика или восстановлений после сбоев
- 03 устанавливаем параллелизм оператора, *если это явно необходимо*
- 04 иначе не переусложняем пайплайн

KEYED STREAM



JOB GRAPH

```
env.fromSource(eventsSource, WatermarkStrategy.noWatermarks(), "events")  
  .name("Source events")  
  .uid("source-events")  
  .setParallelism(3)
```

```
.filter(e -> e.getUid() != null)  
.name("Filter by uid")  
.uid("filter-by-uid")
```

```
.keyBy(Event::getUid)  
.process(new CounterFunction())  
.name("Count events")  
.uid("count-events")
```

```
.sinkTo(eventsSink)  
.name("Sink result")  
.uid("sink-result");
```



NON-KEYED STREAM

- ▶ для `KafkaSource` JobManager назначает партиции, которые слушает каждый консьюмер
- ▶ для операторов, которые расположены после других, с тем же параллелизмом данные остаются в тех же сабтасках, что и были (`chaining`)
- ▶ если параллелизм одинаковый и не произошел `chaining` по какой-то причине - данные из `subtask1` оператора 1 пересылаются в `subtask1` оператора 2, 2 - в 2 и т.д. (`FORWARD`)
- ▶ если между операторами поменялся параллелизм, происходит `REBALANCE` - данные перераспределяются `round robin`

ЗАДАЧА: ФИЛЬТРАЦИЯ ПО IP



Не обрабатываем события с IP адресами из черного списка

(Пополняемый) черный список IP адресов

```
public class BlacklistFilterFunction extends BroadcastProcessFunction<Event, String, Event> {

    public static final MapStateDescriptor<String, Boolean> BROADCAST_BLACKLIST_STATE_DESC =
        new MapStateDescriptor<>("blacklists", String.class, Boolean.class);

    @Override
    public void processElement(Event value, ReadOnlyContext ctx, Collector<Event> out) throws Exception {
        if (ctx.getBroadcastState(BROADCAST_BLACKLIST_STATE_DESC).get(value.getIp()) == null) {
            out.collect(value);
        }
    }

    @Override
    public void processBroadcastElement(String value, Context ctx, Collector<Event> out) throws Exception
    {
        ctx.getBroadcastState(BROADCAST_BLACKLIST_STATE_DESC).put(value, true);
    }
}
```



JOB GRAPH

```
env.fromSource(eventsSource, WatermarkStrategy.noWatermarks(), "events")
    .name("Source events")
    .uid("source-events")
    .setParallelism(1)

    .filter(e -> e.getUid() != null)
    .name("Filter by uid")
    .uid("filter-by-uid")

    .connect (
        env.fromSource(blacklistsSource, WatermarkStrategy.noWatermarks(), "blacklists")
            .name("Source blacklists")
            .uid("source-blacklists")

            .setParallelism(1)
            .broadcast(BROADCAST_BLACKLIST_STATE_DESC)
    )

    .process(new BlacklistFilterFunction())
    .name("Filter blacklist ips")
    .uid("filter-blacklist-ips")

    .keyBy(Event::getUid)
    .process(new CounterFunction())
    .name("Count events")
    .uid("count-events")

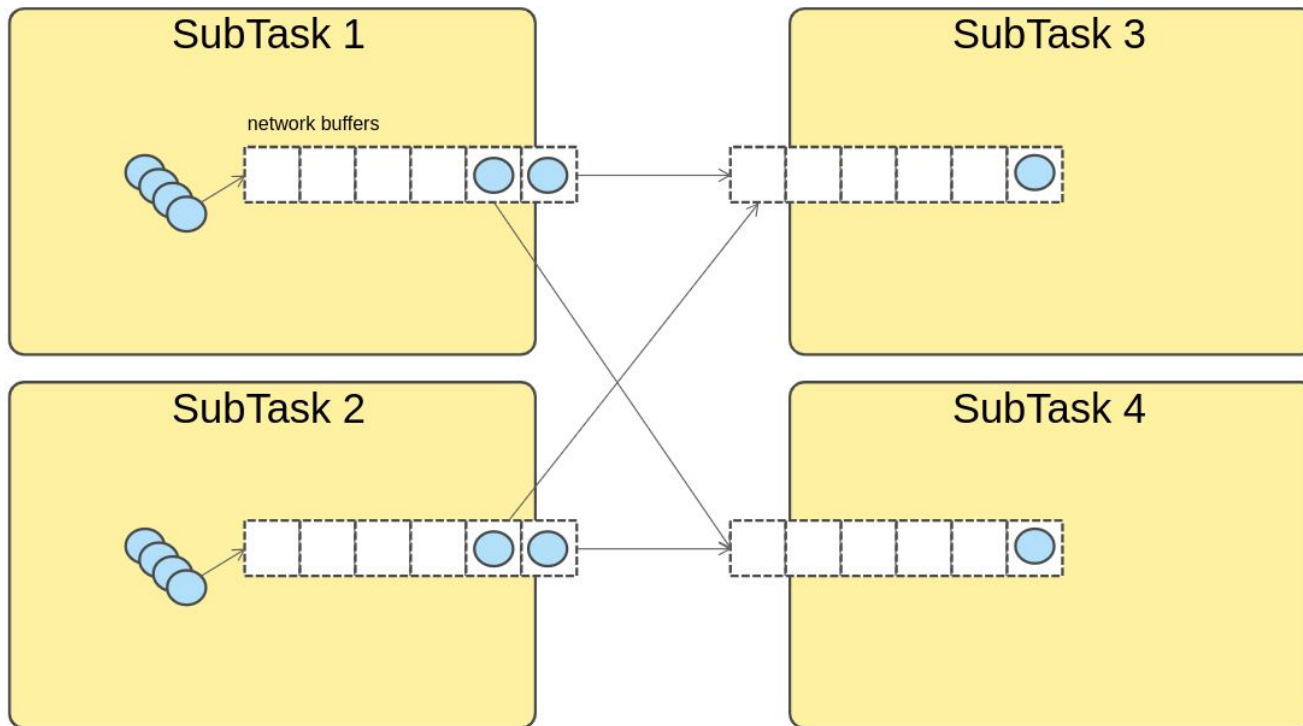
    .sinkTo(eventsSink)
    .name("Sink result")
    .uid("sink-result");
```



CONNECTION TYPES

- ▶ **FORWARD** (прямая, события из одного вышестоящего сабтаска попадают только в один нижестоящий)
- ▶ **HASH** - при партиционировании по ключу
- ▶ **BROADCAST** - события попадают во все сабтаски сразу
- ▶ **REBALANCE** - события распределяются раунд-робином

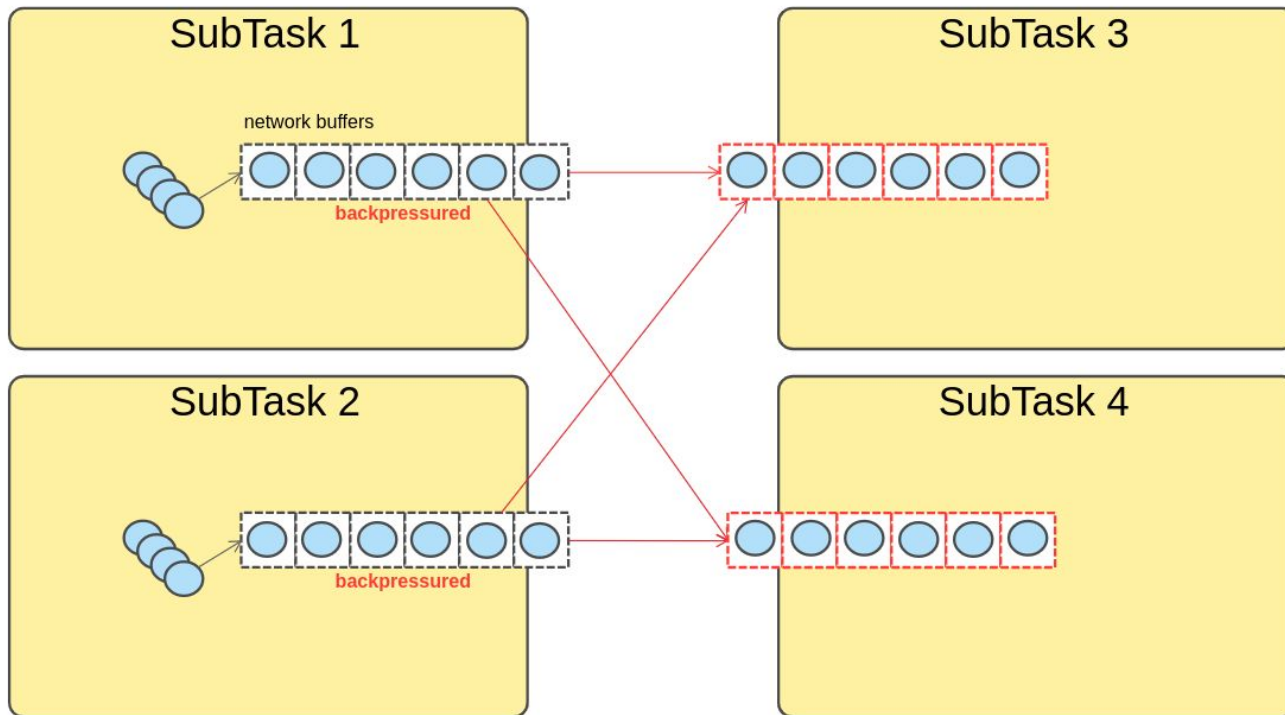
ПЕРЕДАЧА ДАННЫХ ПО СЕТИ



Обратное давление

Материал из Википедии — свободной энциклопедии

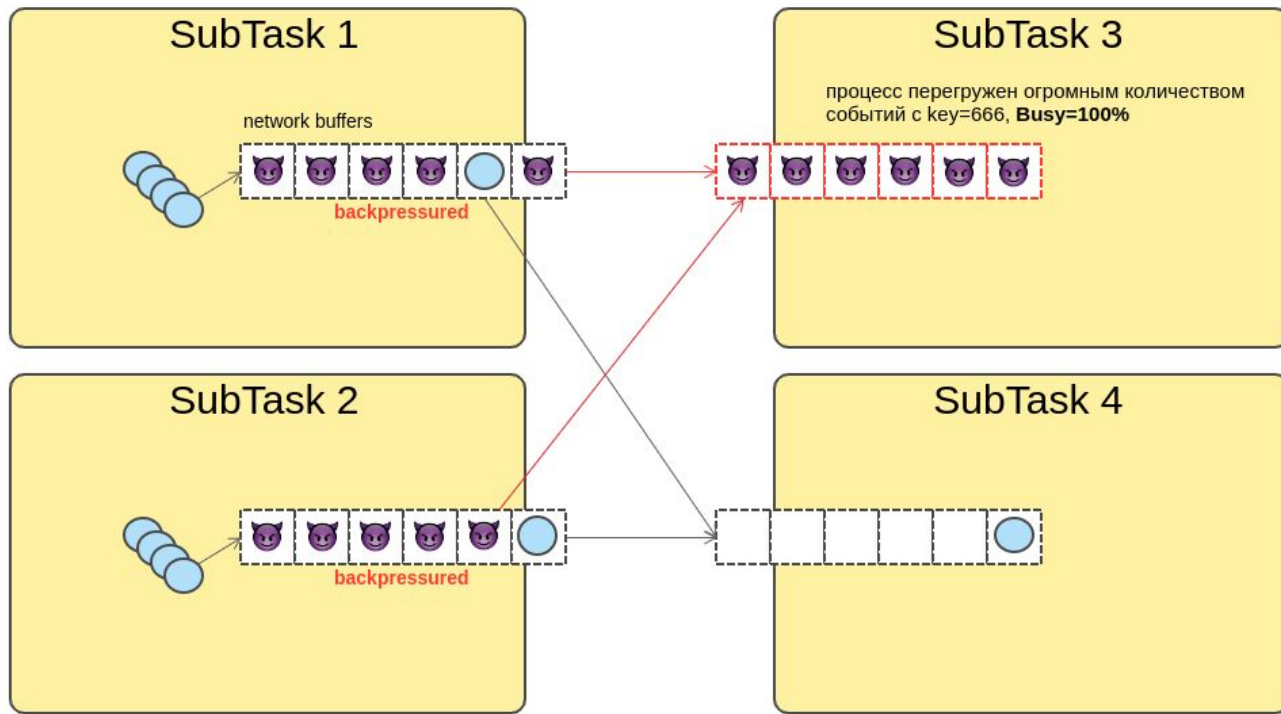
Обратное давление (Back pressure) — это сопротивление или сила, противодействующая желаемому потоку жидкости по трубам, что приводит к потерям на трение и падению давления.



DATA SKEW



пользователь uid=666: парсер/бот
делает 1kk кликов в секунду



ЧТО МЫ ЗНАЕМ О DISTRIBUTED

01 *Как распределяются данные ?*

FORWARD / HASH / REBALANCE / BROADCAST

02 *Как выбрать параллелизм ?*

Эмпирически, по нагрузочным тестам, с запасом и не переусложняя

03 *Отчего бывает Backpressure ?*

Медленная обработка в каких-то из Subtasks: мало ресурсов или Data Skew

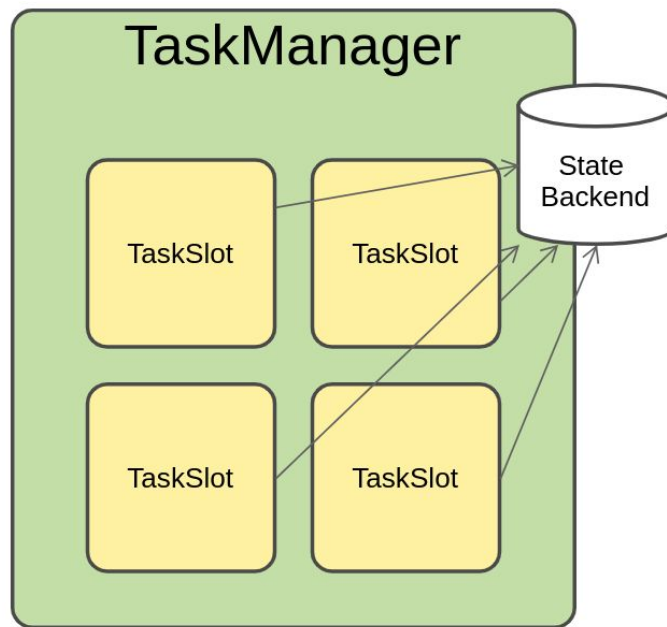
PART II

STATEFUL



STATE BACKEND

- ▶ Стейт хранится в локальном State Backend (HashMap или RocksDB)
- ▶ Локальное хранилище привязано к TaskManager
- ▶ Синхронизации стейта между разными TaskManagers / TaskSlots нет



STATE BACKEND

HASHMAP

- ▶ хранение в Java Heap
- ▶ быстрый доступ к данным
- ▶ объем данных ограничен размером Java Heap
- ▶ нет инкрементальных чекпоинтов

ROCKSDB

- ▶ хранение в файловой системе
- ▶ доступ к данным в общем случае через IO
- ▶ объем данных ограничен объемом дисков
- ▶ есть инкрементальные чекпоинты

ТИПЫ СТЕЙТА

KEYED

стейт привязан к ключу

данные хранятся в key-value хранилище, где ключ партиционирования включен в key

NON-KEYED

стейт привязан только к Subtask

данные сохраняются целиком для всего Subtask

синхронизации между разными Subtask нет.

BROADCAST

стейт одинаковый на всех Subtask из-за того, что копии данных были переданы в каждый Subtask из вышестоящего оператора

синхронизации между Subtask все еще нет

```
public class CounterFunction extends KeyedProcessFunction<Integer, Event, Tuple2<Integer, Integer>> {
```

```
    private ValueState<Integer> counterState;

    @Override
    public void open(Configuration parameters) {
        ValueStateDescriptor<Integer> counterStateDesc = new ValueStateDescriptor<>("counter",
Integer.class);
        counterState = getRuntimeContext().getState(counterStateDesc);
    }

    @Override
    public void processElement(Event value, Context ctx, Collector<Tuple2<Integer, Integer>> out throws
Exception {
        Integer counterValue = counterState.value();
        if (counterValue == null) {
            counterValue = 0;
        }
        counterState.update(++counterValue);
        out.collect(new Tuple2<>(ctx.getCurrentKey(), counterValue));
    }
}
```



```

public class CounterWithTtlFunction extends KeyedProcessFunction<Integer, Event, Tuple2<Integer, Integer>> {

    private ValueState<Integer> counterState;

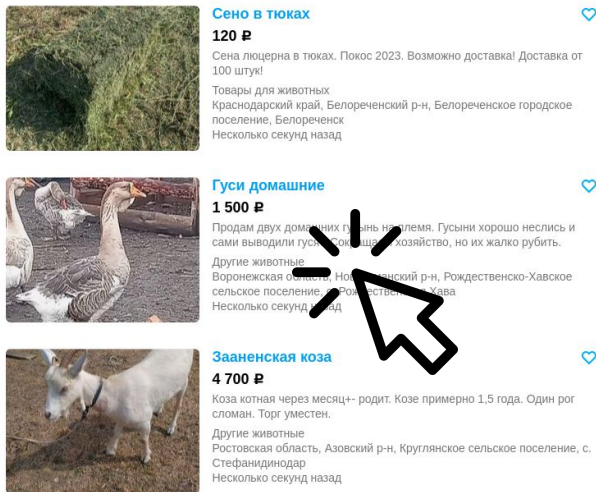
    @Override
    public void open(Configuration parameters) {
        ValueStateDescriptor<Integer> counterStateDesc = new ValueStateDescriptor<>("counter", Integer.class);
        StateTtlConfig valuesTtl = StateTtlConfig.newBuilder(Time.days(1))
            .updateTtlOnReadAndWrite()
            .build();
        counterStateDesc.enableTimeToLive(valuesTtl);
        counterState = getRuntimeContext().getState(counterStateDesc);
    }

    @Override
    public void processElement(Event value, Context ctx, Collector<Tuple2<Integer, Integer>> out) throws Exception
    {
        Integer counterValue = counterState.value();
        if (counterValue == null) {
            counterValue = 0;
        }
        counterState.update(++counterValue);
        out.collect(new Tuple2<>(ctx.getCurrentKey(), counterValue));
    }
}

```



ЗАДАЧА: ДОБАВИТЬ ДЕДУПЛИКАЦИЮ



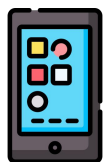
Сено в тюках
120 ₽
Сена люцерны в тюках. Понос 2023. Возможно доставка! Доставка от 100 штук!
Товары для животных
Краснодарский край, Белореченский р-н, Белореченское городское поселение, Белореченск
Несколько секунд назад

Гуси домашние
1 500 ₽
Продам двух домашних гусей на племя. Гусыни хорошо неслись и сами выводили гусят. Живут в хозяйстве, но их жалко рубить.
Другие животные
Воронежская область, Новоаннинский р-н, Рождественско-Хавское сельское поселение, Рождественск-Хавский район, Хавза
Несколько секунд назад

Зааненская коза
4 700 ₽
Коза котная через месяц+- родит. Козе примерно 1,5 года. Один рог сломан. Торг уместен.
Другие животные
Ростовская область, Азовский р-н, Круглянское сельское поселение, с. Стефандинодар
Несколько секунд назад

```
{  
  "id": 1,  
  "uid": 30,  
  "ip": "87.90.89.74",  
  "itemId": 955,  
  "timestamp": 1691482581510  
}
```

На каждое объявление учитываем только один клик



kafka



ВИДЫ СТЕЙТА

Value State

одно значение

всегда десериализуется/сериализуется целиком при чтении и записи.

List State

список значений

быстро работает на запись (весь список не десериализуется для добавления элемента)

при чтении всегда десериализуется целиком

Map State

мапа

быстро работает и на чтение, и на запись

данные десериализуются/сериализуются только для нужного ключа

```
private ListState<Integer> itemsState;
```

```
@Override
```

```
public void processElement(Event value, Context ctx, Collector<Tuple2<Integer, Integer>> out) throws
```

```
Exception {
```

```
    Integer counterValue = counterState.value();
```

```
    if (counterValue == null) {
```

```
        counterValue = 0;
```

```
    }
```

```
    Integer itemId = value.getItemId();
```

```
    if(itemId != null && itemId > 0) {
```

```
        for(Integer iid: itemsState.get()) {
```

```
            if(itemId.equals(iid)) {
```

```
                return;
```

```
            }
```

```
        }
```

```
    }
```

```
    itemsState.add(itemId);
```

```
    counterState.update(++counterValue);
```

```
    out.collect(new Tuple2<>(ctx.getCurrentKey(), counterValue));
```

```
}
```



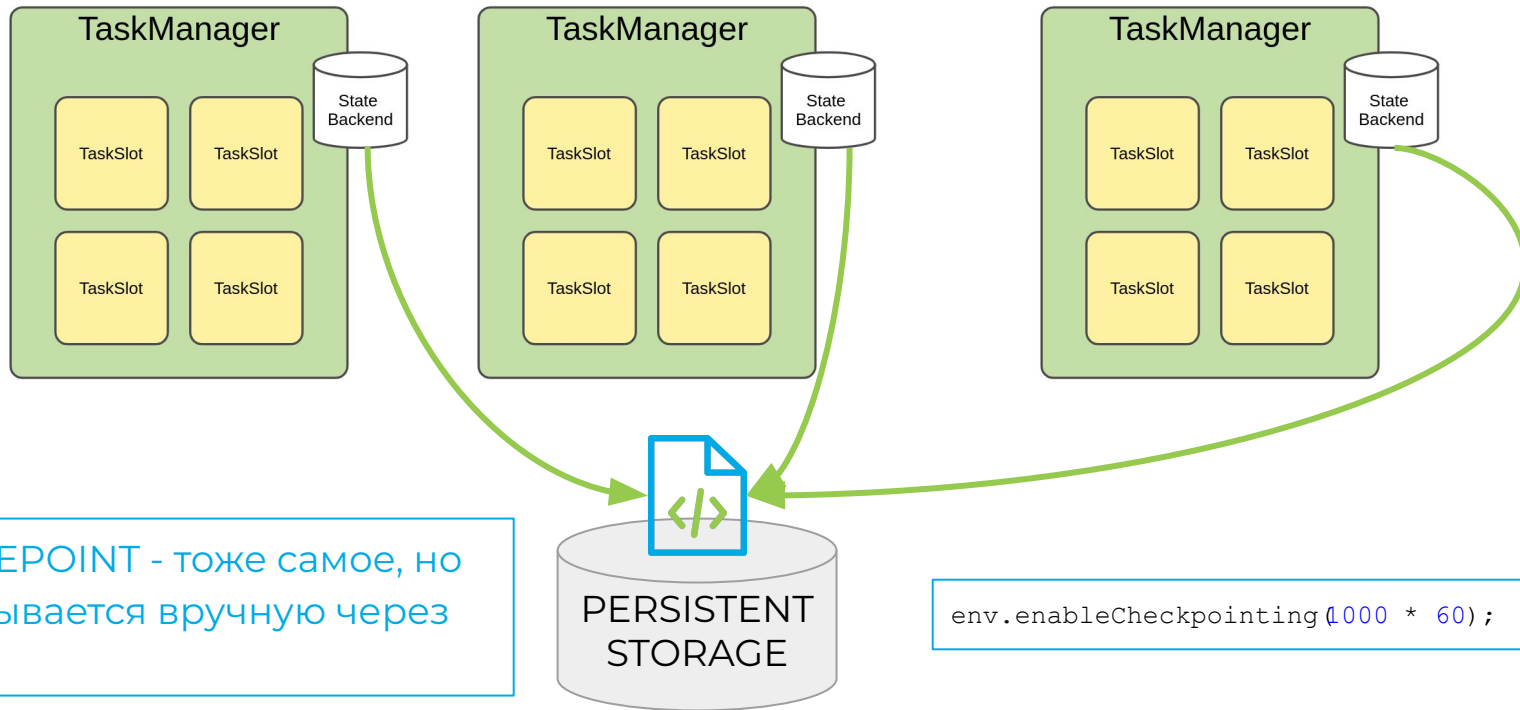
```
private MapState<Integer, Boolean> itemsState;

@Override
public void processElement(Event value, Context ctx, Collector<Tuple2<Integer, Integer>> out) throws
Exception {
    Integer counterValue = counterState.value();
    if (counterValue == null) {
        counterValue = 0;
    }

    Integer itemId = value.getItemId();
    if(itemId != null && itemId > 0 && itemsState.contains(itemId)) {
        return;
    }
    itemsState.put(itemId, true);
    counterState.update(++counterValue);
    out.collect(new Tuple2<>(ctx.getCurrentKey(), counterValue));
}
```



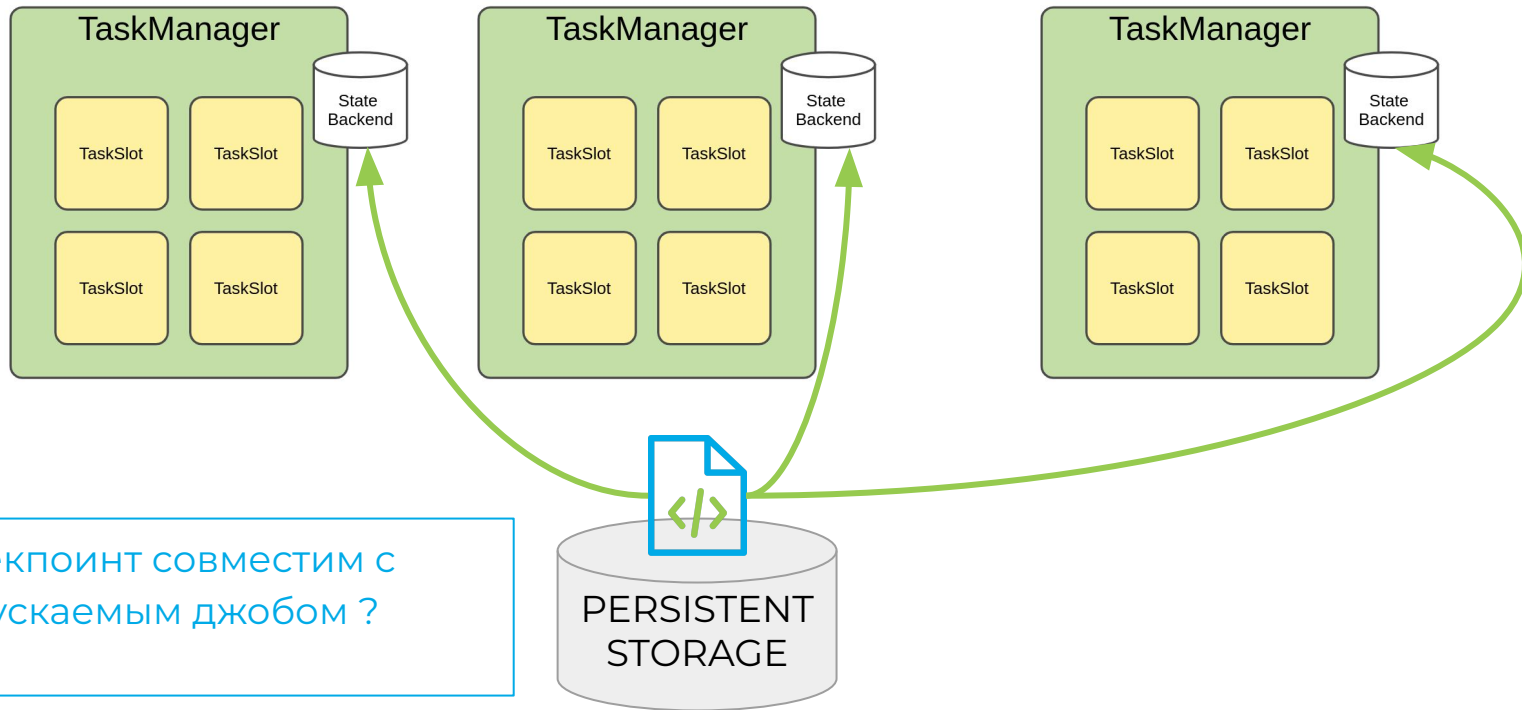
PERIODICAL CHECKPOINT



SAVEPOINT - тоже самое, но
вызывается вручную через
API

```
env.enableCheckpointing(1000 * 60);
```

RESTORE JOB



А чекпоинт совместим с запускаемым джобом ?

ЭВОЛЮЦИЯ СТЕЙТА

Что делать, если структура стейта изменилась ?

- ▶ Стейт для оператора подтягивается по uid оператора и id стейта
- ▶ Если оператор был удален из джобы или стейт был удален из оператора, стейт больше не нужен: при запуске нужно указать флаг Allow Non Restored State
- ▶ Добавление или удаление полей в классе стейта допустимо
- ▶ При изменении типа стейта или типа поля стейта эволюция невозможна
- ▶ Если у стейта не был задан TTL, а затем он появился, эволюция невозможна
- ▶ Стейт с типом, сериализуемым KryoSerializer, эволюционировать не может

ЧТО МЫ ЗНАЕМ О STATEFUL

01 *Где хранится стейт ?*

В State Backend - HashMap или RocksDB (но в проде лучше RocksDB), а также в чекпоинтах в персистентном хранилище

02 *Как хранится стейт ?*

В структурах ValueState / ListState / MapState, а сами объекты (в RocksDB) - в сериализованном виде

03 *На что обратить внимание ?*

Не допускать избыточной сериализации/десериализации, при изменениях в стейте соблюдать правила эволюции типов, чтобы не потерять данные


PART III

REALTIME

ЗАДАЧА: ПЕРИОДИЧЕСКАЯ ОТПРАВКА



Сено в тюках
120 Р
Сена люцерна в тюках. Покос 2023. Возможно доставка! Доставка от 100 штук!
Товары для животных
Краснодарский край, Белореченский р-н, Белореченское городское поселение, Белореченск
Несколько секунд назад

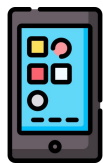


Гуси домашние
1 500 Р
Продам двух домашних гусей на племя. Гусыни хорошо неслись и сами выводили гусят. Живут в хозяйстве, но их жалко рубить.
Другие животные
Воронежская область, Новоаннинский р-н, Рождественско-Хавское сельское поселение, Рождественск-Хавский район
Несколько секунд назад



Зааненская коза
4 700 Р
Коза котная через месяц+- родит. Козе примерно 1,5 года. Один рог сломан. Торг уместен.
Другие животные
Ростовская область, Азовский р-н, Круглянское сельское поселение, с. Стефандинодар
Несколько секунд назад

отправляем результаты спустя 10 минут после первого клика пользователя, далее каждые 10 минут

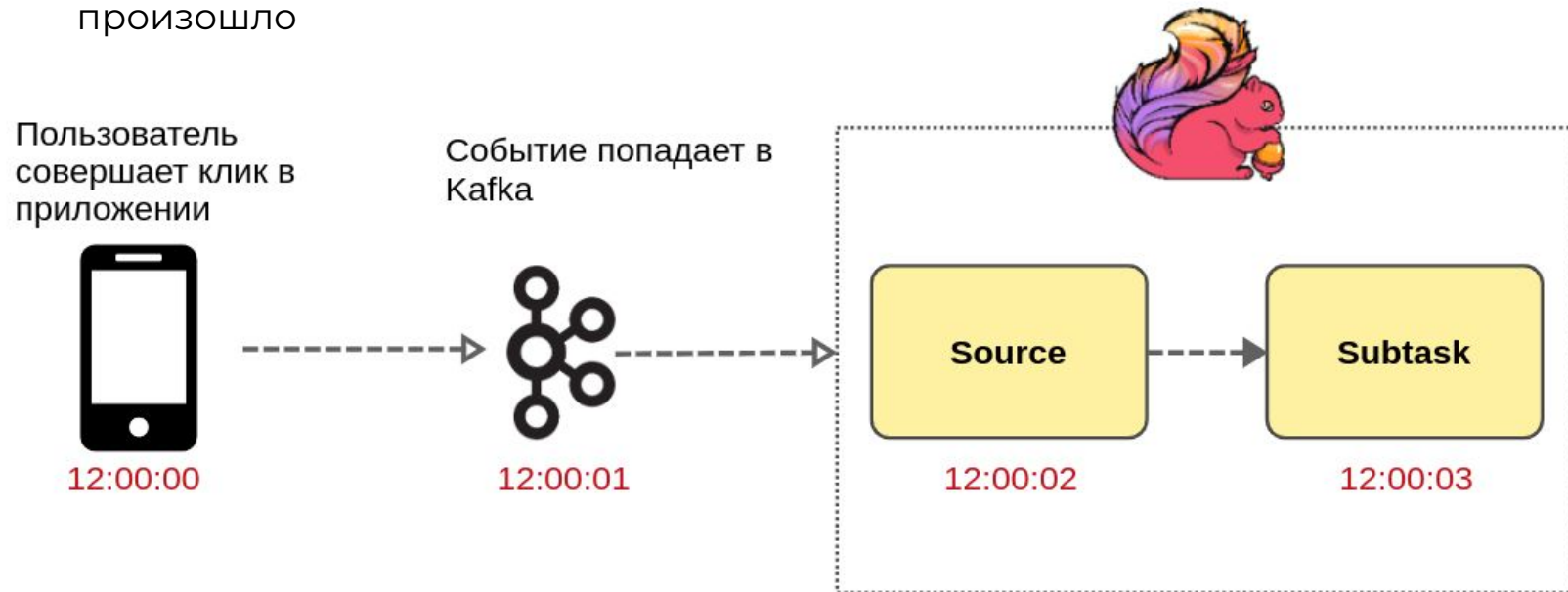


kafka



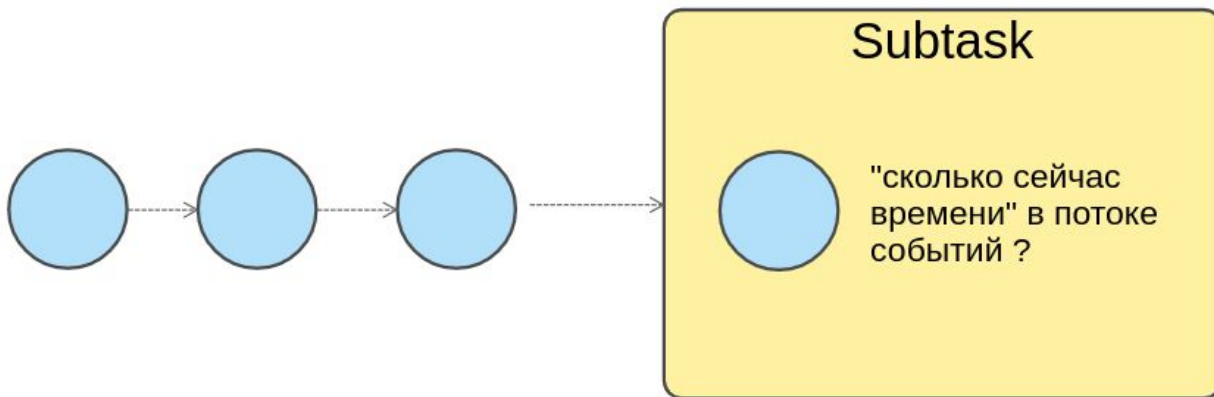
ЧТО ТАКОЕ ВРЕМЯ (СОБЫТИЯ) ?

- ▶ События не обрабатываются мгновенно, проходит время на доставку события от пользователя до обработки
- ▶ Несмотря на то, что наша система Realtime, время обработки события отличается от времени, когда событие реально произошло



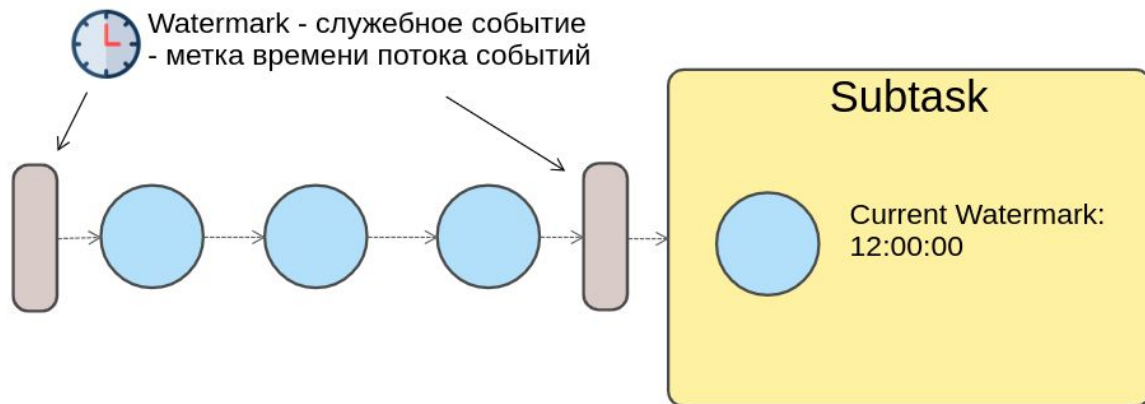
НОТАЦИЯ ВРЕМЕНИ

- ▶ **Processing Time** - текущее серверное время обработки события
- ▶ **Event Time** - фактическое время события. Что такое **текущее** Event Time ?



WATERMARK

- ▶ **Watermark** показывает текущее время в потоке
- ▶ Значение воте́рмарки вычисляется из **времени событий** (значение поля timestamp)
- ▶ По умолчанию периодическая воте́рмарка генерируется **на source** каждые 200 мс.



Инициализируем вотермарку

```
env.fromSource(eventsSource,  
    WatermarkStrategy.<Event>forBoundedOutOfOrderness(Duration.ofMinutes(1))  
        .withTimestampAssigner((e, ts) -> e.getTimestamp()),  
    "events"  
)  
.name("Source events")  
.uid("source-events")
```

- ▶ вотермарка определяется как **максимальное** время Event Time из всех встреченных событий - 1 минута
- ▶ считаем, что в общем случае события **не опаздывают** более, чем на 1 минуту



```

@Override
public void processElement(Event value, Context ctx, Collector<Tuple2<Integer, Integer>> out) throws Exception {
    Integer counterValue = counterState.value();
    if (counterValue == null) {
        counterValue = 0;
        ctx.timerService().registerEventTimeTimer(value.getTimestamp() + 1000 * 60 * 10);
    }

    Integer itemId = value.getItemId();
    if(itemId != null && itemId > 0 && itemsState.contains(itemId)) {
        return;
    }
    itemsState.put(itemId, true);
    counterState.update(++counterValue);
}

```

```

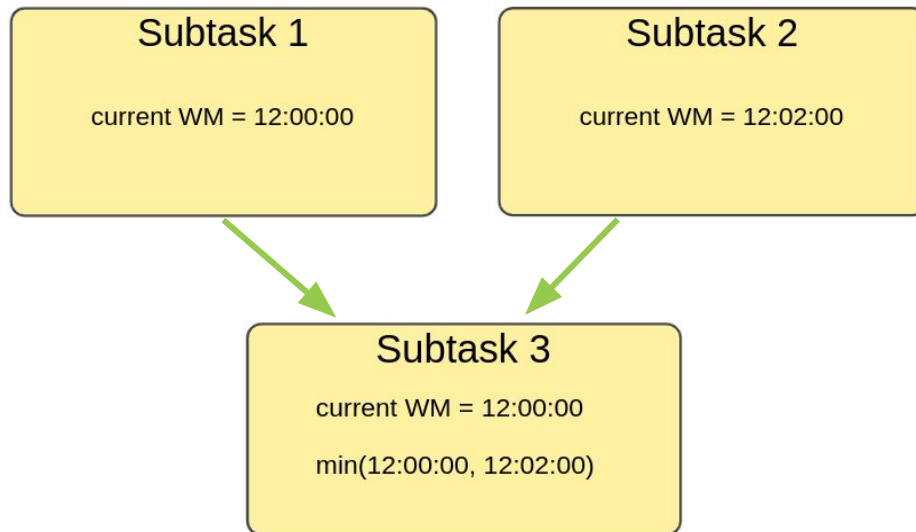
@Override
public void onTimer(long timestamp, OnTimerContext ctx, Collector<Tuple2<Integer, Integer>> out) throws Exception {
    Integer counterValue = counterState.value();
    if(counterValue == null || counterValue == 0) {
        return;
    }
    out.collect(new Tuple2<>(ctx.getCurrentKey(), counterValue));
    ctx.timerService().registerEventTimeTimer(timestamp + 1000 * 60 * 10);
    counterState.update(0);
    itemsState.clear();
}

```



ВОТЕРМАРКА МЕЖДУ ПОТОКАМИ

- ▶ **output** watermark - вотермарка “на выходе” из Subtask
- ▶ **input** watermark - вотермарка “на входе” в Subtask
- ▶ вотермарка в каждом Subtask определяется как **МИНИМАЛЬНАЯ** из всех входящих вотермарок



Инициализируем вотермарку

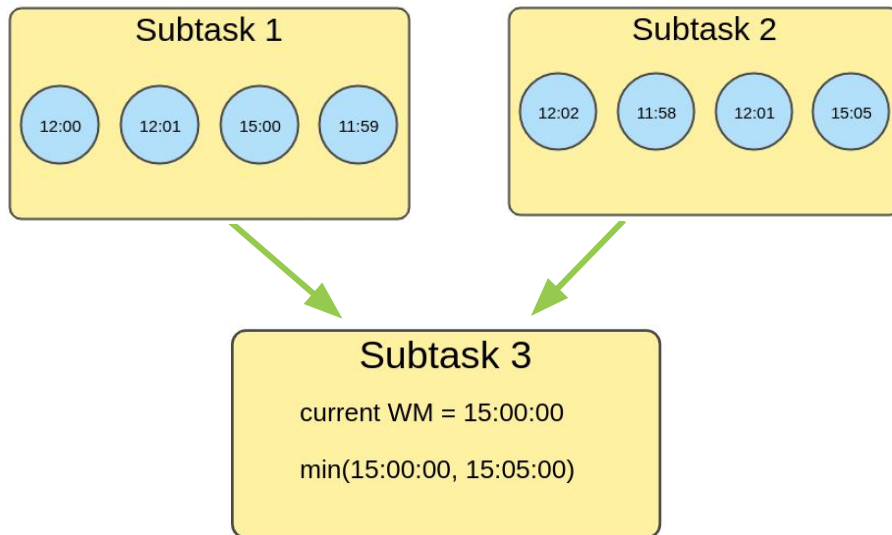
```
env.fromSource(blacklistsSource,  
    WatermarkStrategy.<String>noWatermarks().withIdleness(Duration.ofSeconds(1)),  
    "blacklists"  
)  
.name("Source blacklists")  
.uid("source-blacklists")
```

- ▶ для blacklist вотермарка не имеет смысла (*noWatermarks*)
- ▶ через 1 секунду тишины из топика blacklists вотермарка из этого источника перестанет учитываться в качестве input watermark



СОБЫТИЯ ИЗ БУДУЩЕГО

- ▶ даже небольшое количество событий из будущего, попавших во все Subtasks на источнике, сдвинут вотермарку в будущее
- ▶ после исправления ситуации при перезапуске джобы вотермарка пересчитается заново



ЧТО МЫ ЗНАЕМ О REALTIME

01 *Что такое время ?*

Есть две нотации - Processing Time (текущее время) и Event Time (реальное время, когда произошло событие)

02 *Что такое вотермарка ?*

Вотермарка - служебное событие, показывающее, “сколько сейчас времени” в потоке по Event Time

03 *На что обратить внимание ?*

Вотермарка при передаче событий между задачами складывается из всех входящих вотермарок. События из будущего могут испортить логику работы и это нельзя будет исправить без перезапуска

avito.tech

Спасибо !

Весь код из примеров можно [найти на github](#)

