YTsaurus

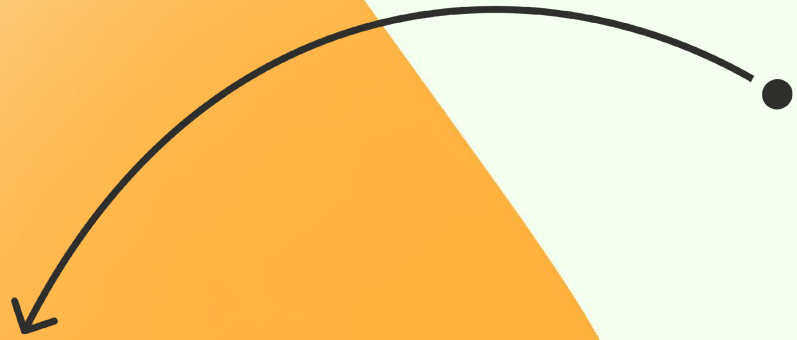
Roren — C++ фреймворк для описания пайплайнов распределенных вычислений

Дмитрий Ермолов,
Яндекс

24.05

План презентации

1. Модели распределенных систем
2. Проблемы существовавших SDK
3. Основные концепции Roren
4. Идеи реализации: исполнение пользовательского кода на удаленных машинах



Модели распределенных систем

MapReduce и Streaming

MapReduce и Streaming

Процессы описываются набором
простых кирпичиков

Инфраструктура даёт

- Горизонтальное масштабирование
- Отказоустойчивость

MapReduce

Пакетная обработка больших таблиц

Пример процесса

- Дано: данные о запросах пользователей
- Задача: построить профиль интересов пользователей

Query	UserId
Умные указатели	1412
std::format	1412
Краска для мебели	1412

UserId	UserProfile
1412	{"C++": 2, "ремонт": 1}

MapReduce: map

Query	UserId
Умные указатели	1412
Детские кроссовки	2109
std::format	1412
Краска для мебели	1412
Флоксы	2109
Беговел	2109

Category	UserId
C++	1412
Дети	2109
C++	1412
Ремонт	1412
Сад	2109
Дети	2109

MapReduce: shuffle

Category	UserId
C++	1412
Дети	2109
C++	1412
Ремонт	1412
Сад	2109
Дети	2109

UserId	Category
2109	Дети
2109	Сад
2109	Дети
1412	C++
1412	C++
1412	Ремонт

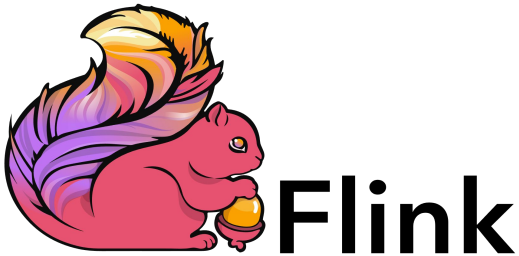
MapReduce: reduce

UserId	Category
2109	Дети
2109	Сад
2109	Дети
1412	C++
1412	C++
1412	Ремонт

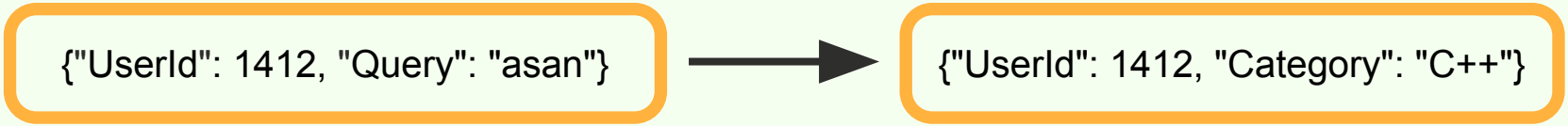
UserId	Category
2109	{"дети": 2, "сад": 1}
1412	{"C++": 2, "ремонт": 1}

Streaming

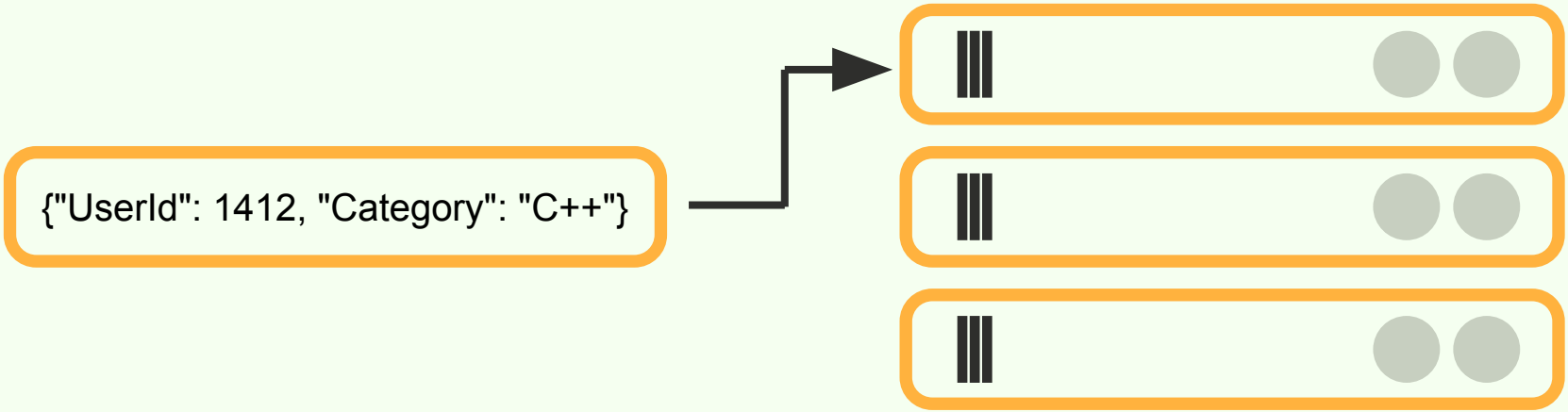
Обработка потоков данных в режиме
(около) реального времени



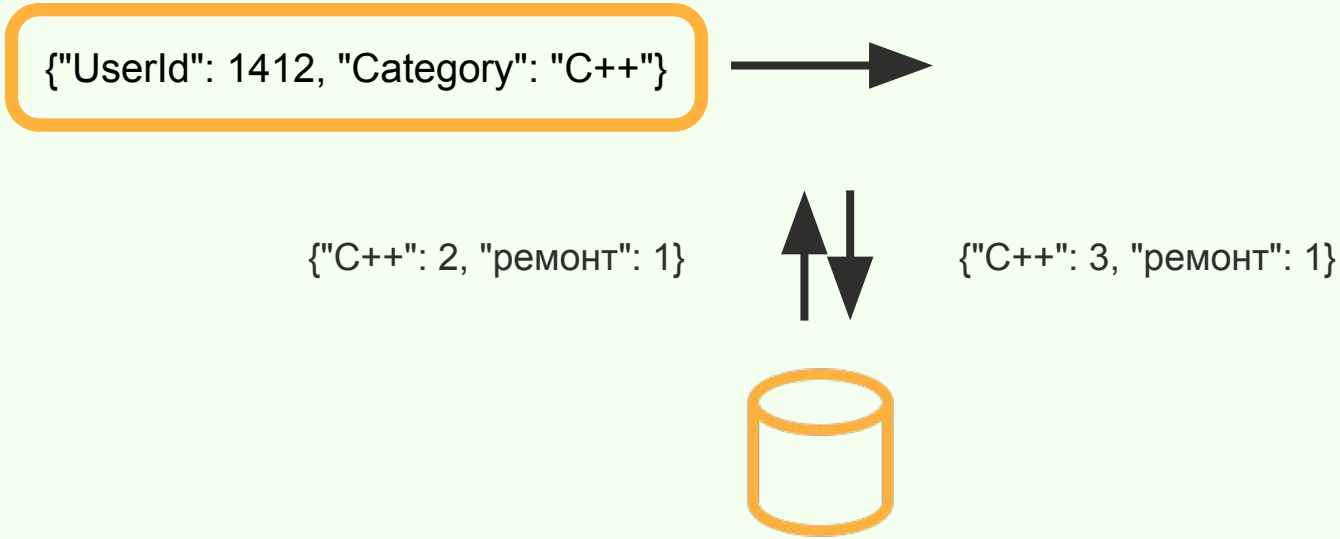
Streaming: map



Streaming: shuffle



Streaming: stateful map



Проблемы существовавших SDK

MapReduce

```
// API в императивном стиле
client->MapReduce("//input", "//tmp", make_unique<TMapper1>(), make_unique<TReducer1>());
client->MapReduce("//tmp", "//output", make_unique<TMapper2>(), make_unique<TReducer2>());

// Неудобно описывать pipeline'ы вычислений
```

MapReduce

```
// API в императивном стиле
client->MapReduce("//input", "//tmp", make_unique<TMapper1>(), make_unique<TReducer1>());
client->MapReduce("//tmp", "//output", make_unique<TMapper2>(), make_unique<TReducer2>());

// Неудобно описывать pipeline'ы вычислений:
//   1. Нужно управлять временными таблицами
```

MapReduce

```
// API в императивном стиле
client->MapReduce("//input", "//tmp", make_unique<TMapper1>(), make_unique<TReducer1>());
client->MapReduce("//tmp", "//output", make_unique<TMapper2>(), make_unique<TReducer2>());

// Неудобно описывать pipeline'ы вычислений:
//   1. Нужно управлять временными таблицами
//   2. Нужно отслеживать, когда можно запустить очередную операцию
```

MapReduce

```
// Один класс на процесс,
//   стадии соединяются через конфиги
class TProcessor : public IProcessor {
    void Process(TMessageBatch &messageBatch) override { ... }
};
```

MapReduce

```
// Один класс на процесс,
//   стадии соединяются через конфиги
class TProcessor : public IProcessor {
    void Process(TMessageBatch &messageBatch) override { ... }
};

// 1. Вообще нет возможности описывать pipeline'ы
```

MapReduce

```
// Один класс на процесс,
//   стадии соединяются через конфиги
class TProcessor : public IProcessor {
    void Process(TMessageBatch &messageBatch) override { ... }
};

// 1. Вообще нет возможности описывать pipeline'ы
// 2. Хочется уметь запускать один код и в MapReduce, и в Streaming
```

Возможные решения

YQL — SQL
для распределённых систем

SQL и C++ неудобно интегрировать

Apache Spark,
Apache Beam

Java, Python

Roren: основные концепции

MapReduce

```
auto pipeline = MakeYtPipeline( "some.cluster.ytsaurus.tech" );

TPCollection<TQueryLogProto> logs = pipeline | YtRead< TQueryLogProto >("//logs/queries/2023-05-24" );

TPCollection<TKV< int, std::string>> userToCategory = logs | ParDo([] ( const TQueryLogProto & log) {
    return TKV<int, std::string>{log.user_id(), ResolveCategory(log.query())};
});

TPState<int, TUserProfileProto> pState = MakeYtPState(std::nullopt,   "//profiles/2023-05-24" );
userToCategory | StatefulParDo(pState,
    [] (const TKV<int, std::string>& input, TOutput<void>&, TUserProfileProto & profile) {
        (*userProfile.mutable_interests())[input.Value()]++;
});

pipeline.Run();
```

MapReduce

```
auto pipeline = MakeYtPipeline( "some.cluster.ytsaurus.tech" ); // Хранит граф вычислений

TPCollection<TQueryLogProto> logs = pipeline | YtRead<TQueryLogProto>("//logs/queries/2023-05-24");

TPCollection<TKV<int, std::string>> userToCategory = logs | ParDo([] (const TQueryLogProto& log) {
    return TKV<int, std::string>{log.user_id(), ResolveCategory(log.query())};
});

TPState<int, TUserProfileProto> pState = MakeYtPState(std::nullopt, "//profiles/2023-05-24");
userToCategory | StatefulParDo(pState,
    [] (const TKV<int, std::string>& input, TOutput<void>&, TUserProfileProto& profile) {
        (*userProfile.mutable_interests())[input.Value()]++;
});

pipeline.Run(); // Может запустить вычисления на выбранной системе
```

MapReduce

```
auto pipeline = MakeYtPipeline("some.cluster.ytsaurus.tech");

TPCollection<TQueryLogProto>  logs = pipeline | YtRead<TQueryLogProto>("//logs/queries/2023-05-24");

TPCollection<TKV< int, std::string>> userToCategory = logs | ParDo([] (const TQueryLogProto& log) {
    return TKV<int, std::string>{log.user_id(), ResolveCategory(log.query())};
});

TPState<int, TUserProfileProto> pState = MakeYtPState(std::nullopt, "//profiles/2023-05-24");
userToCategory | StatefulParDo(pState,
    [] (const TKV<int, std::string>& input, TOutput<void>&, TUserProfileProto& profile) {
        (*userProfile.mutable_interests())[input.Value()]++;
});

pipeline.Run();
```

MapReduce

```
auto pipeline = MakeYtPipeline("some.cluster.ytsaurus.tech");

TPCollection<TQueryLogProto>  logs = pipeline | YtRead<TQueryLogProto>("//logs/queries/2023-05-24");

TPCollection<TKV< int, std::string>>  userToCategory = logs | ParDo([] (const TQueryLogProto& log) {

// Свойства PCollection (PCollection от Parallel Collection):
```

MapReduce

```
auto pipeline = MakeYtPipeline("some.cluster.ytsaurus.tech");

TPCollection<TQueryLogProto>  logs = pipeline | YtRead<TQueryLogProto>("//logs/queries/2023-05-24");

TPCollection<TKV< int, std::string>>  userToCategory = logs | ParDo([] (const TQueryLogProto& log) {

// Свойства PCollection (PCollection от Parallel Collection):
//   1. Нельзя узнать свойства коллекции
```

MapReduce

```
auto pipeline = MakeYtPipeline("some.cluster.ytsaurus.tech");

TPCollection<TQueryLogProto>  logs = pipeline | YtRead<TQueryLogProto>("//logs/queries/2023-05-24");

TPCollection<TKV< int, std::string>>  userToCategory = logs | ParDo([] (const TQueryLogProto& log) {

// Свойства PCollection (PCollection от Parallel Collection):
//   1. Нельзя узнать свойства коллекции
//   2. Нельзя читать или писать элементы
```

MapReduce

```
auto pipeline = MakeYtPipeline("some.cluster.ytsaurus.tech");

TPCollection<TQueryLogProto>  logs = pipeline | YtRead<TQueryLogProto>("//logs/queries/2023-05-24");

TPCollection<TKV<int, std::string>> userToCategory = logs | ParDo([] (const TQueryLogProto& log) {

// Свойства PCollection (PCollection от Parallel Collection):
//   1. Нельзя узнать свойства коллекции
//   2. Нельзя читать или писать элементы
//   3. Можно применять PTransform'ы, получая новые коллекции
```

MapReduce

```
auto pipeline = MakeYtPipeline("some.cluster.ytsaurus.tech");

TPCollection<TQueryLogProto> logs = pipeline  | YtRead<TQueryLogProto>("//logs/queries/2023-05-24");

TPCollection<TKV<int, std::string>> userToCategory = logs  | ParDo([] (const TQueryLogProto& log) {
    return TKV<int, std::string>{log.user_id(), ResolveCategory(log.query())};
});

TPState<int, TUserProfileProto> pState = MakeYtPState(std::nullopt, "//profiles/2023-05-24");
userToCategory | StatefulParDo(pState,
    [] (const TKV<int, std::string>& input, TOutput<void>&, TUserProfileProto& profile) {
        (*userProfile.mutable_interests())[input.Value()]++;
});

pipeline.Run();
```

MapReduce

```
auto pipeline = MakeYtPipeline("some.cluster.ytsaurus.tech");

TPCollection<TQueryLogProto> logs = pipeline   | YtRead<TQueryLogProto>("//logs/queries/2023-05-24");

TPCollection<TKV<int, std::string>> userToCategory = logs   | ParDo([] (const TQueryLogProto& log) {
    return TKV<int, std::string>{log.user_id(), ResolveCategory(log.query())};
});

TPState<int, TUserProfileProto> pState = MakeYtPState(std::nullopt, "//profiles/2023-05-24");
userToCategory  | StatefulParDo(pState,

// 1. Преобразуют PCollection'ы
```

MapReduce

```
auto pipeline = MakeYtPipeline("some.cluster.ytsaurus.tech");

TPCollection<TQueryLogProto> logs = pipeline   | YtRead<TQueryLogProto>("//logs/queries/2023-05-24");

TPCollection<TKV<int, std::string>> userToCategory = logs   | ParDo([] (const TQueryLogProto& log) {
    return TKV<int, std::string>{log.user_id(), ResolveCategory(log.query())};
});

TPState<int, TUserProfileProto> pState = MakeYtPState(std::nullopt, "//profiles/2023-05-24");
userToCategory  | StatefulParDo(pState,

// 1. Преобразуют PCollection'ы
// 2. Применение через оператор pipe (аналогично std::ranges или shell)
```

MapReduce

```
auto pipeline = MakeYtPipeline("some.cluster.ytsaurus.tech");

TPCollection<TQueryLogProto> logs = pipeline   | YtRead<TQueryLogProto>("//logs/queries/2023-05-24");

TPCollection<TKV<int, std::string>> userToCategory = logs   | ParDo([] (const TQueryLogProto& log) {
    return TKV<int, std::string>{log.user_id(), ResolveCategory(log.query())};
});

TPState<int, TUserProfileProto> pState = MakeYtPState(std::nullopt, "//profiles/2023-05-24");
userToCategory  | StatefulParDo(pState,

// 1. Преобразуют PCollection'ы
// 2. Применение через оператор pipe (аналогично std::ranges или shell)
// 3. В момент применения никакой работы не происходит
```

MapReduce

```
auto pipeline = MakeYtPipeline("some.cluster.ytsaurus.tech");

TPCollection<TQueryLogProto> logs = pipeline | YtRead< TQueryLogProto >("//logs/queries/2023-05-24" );

TPCollection<TKV<int, std::string>> userToCategory = logs | ParDo([] (const TQueryLogProto& log) {
    return TKV<int, std::string>{log.user_id(), ResolveCategory(log.query())};
});

TPState<int, TUserProfileProto> pState = MakeYtPState(std::nullopt, "//profiles/2023-05-24");
userToCategory | StatefulParDo(pState,
    [] (const TKV<int, std::string>& input, TOutput<void>&, TUserProfileProto& profile) {
        (*userProfile.mutable_interests())[input.Value()]++;
});

pipeline.Run();
```

MapReduce

```
auto pipeline = MakeYtPipeline("some.cluster.ytsaurus.tech");

TPCollection<TQueryLogProto> logs = pipeline | YtRead<TQueryLogProto>("//logs/queries/2023-05-24");

TPCollection<TKV<int, std::string>> userToCategory = logs | ParDo([] ( const TQueryLogProto& log) {
    return TKV<int, std::string>{log.user_id(), ResolveCategory(log.query())};
});

TPState<int, TUserProfileProto> pState = MakeYtPState(std::nullopt, "//profiles/2023-05-24");
userToCategory | StatefulParDo(pState,
    [] (const TKV<int, std::string>& input, TOutput<void>&, TUserProfileProto& profile) {
        (*userProfile.mutable_interests())[input.Value()]++;
});

pipeline.Run();
```

MapReduce

```
auto pipeline = MakeYtPipeline("some.cluster.ytsaurus.tech");

TPCollection<TQueryLogProto> logs = pipeline | YtRead<TQueryLogProto>("//logs/queries/2023-05-24");

TPCollection<TKV<int, std::string>> userToCategory = logs | ParDo([] (const TQueryLogProto& log) {
    return TKV<int, std::string>{log.user_id(), ResolveCategory(log.query())};
});

TPState<int, TUserProfileProto> pState = MakeYtPState(std::nullopt,   "//profiles/2023-05-24" );
userToCategory | StatefulParDo(pState,
    [] (const TKV<int, std::string>& input, TOutput<void>&, TUserProfileProto & profile) {
        (*userProfile.mutable_interests())[input.Value()]++;
});

pipeline.Run();
```

MapReduce

```
auto pipeline = MakeYtPipeline("some.cluster.ytsaurus.tech");

TPCollection<TQueryLogProto> logs = pipeline | YtRead<TQueryLogProto>("//logs/queries/2023-05-24");

TPCollection<TKV<int, std::string>> userToCategory = logs | ParDo([] (const TQueryLogProto& log) {
    return TKV<int, std::string>{log.user_id(), ResolveCategory(log.query())};
});

TPState<int, TUserProfileProto> pState = MakeYtPState(std::nullopt,   "//profiles/2023-05-24" );
userToCategory | StatefulParDo(pState,
    [] (const TKV<int, std::string>& input, TOutput<void>&, TUserProfileProto & profile) {
        (*userProfile.mutable_interests())[input.Value()]++;
});

pipeline.Run();
```

MapReduce

```
auto pipeline = MakeYtPipeline( "some.cluster.ytsaurus.tech" );

auto pState = MakeYtPState(std::nullopt,  "//profiles/2023-05-24" );
pipeline
   | YtRead<TQueryLogProto>("//logs/queries/2023-05-24" );
   | ParDo([] (const TQueryLogProto& log) {
       return TKV<int, std::string>{log.user_id(), ResolveCategory(log.query())};
   });
   | StatefulParDo(pState,
       [](const TKV<int, std::string> &input, TOutput<void> &, TUserProfileProto  &profile) {
           (*userProfile.mutable_interests())[input.Value()]++;
       });

pipeline.Run();  // Уже решили часть проблем
```

Что удалось сделать

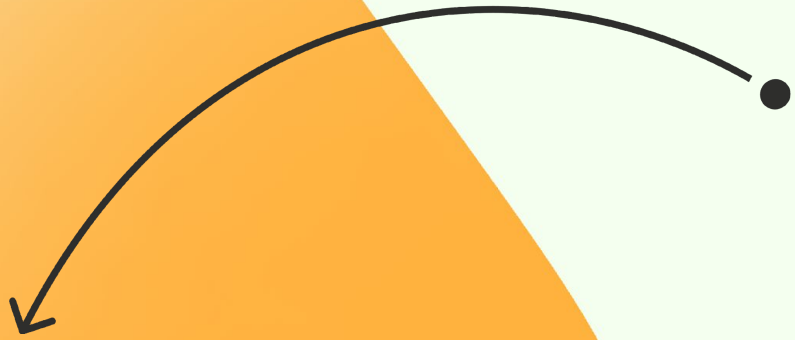
- Код можно запускать на BigRT, YT или локально

Что удалось сделать

- Код можно запускать на BigRT, YT или локально
- Улучшили переиспользование кода

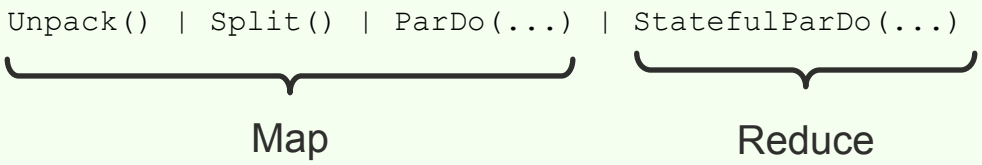
```
// Было
for (auto& message : messageBatch) {
  if (message.Unpack()) ++Metric("UnpackOk");
  else ++Metric("UnpackError");
  TSplitter splitter(message); std::string_view eventBytes;
  while (splitter.Next(&eventBytes)) { ... }
}
// Стало
ReadMessageBatch() | Unpack() | Split() | ...
```

Roren: идеи реализации



pipeline.Run()

- Оптимизирует и отображает PTransform'ы на базовые операции системы
- Запускает исполнение на рабочих машинах



Доставка кода

Доставка кода

- Копируем бинарник

Доставка кода

- Копируем бинарник
- Только Linux

Доставка кода

- Копируем бинарник
- Только Linux
- Статическая линковка

Локализация кода

```
logs | ParDo([] (const TQueryLogEntry& logEntry) {
   auto category = ResolveCategory(logEntry.query());
   return TKV<int, std::string>{logEntry.user_id(), category};
});
```

Локализация кода

```
logs | ParDo([] (const TQueryLogEntry& logEntry) {
    auto category = ResolveCategory(logEntry.query());
    return TKV<int, std::string>{logEntry.user_id(), category};
});

...
auto Apply(const TCollection<T>& pCollection, const F& f) {
    if constexpr (IsAppliable<F, T>) {
        // Этот указатель будет отправлен на сервер, как число.
        // Так можно передать на сервер функцию и ничего не захватывающую лямбду.
        R (*)(const T&) fPtr = f;
        ...
    }
}
```

Локализация кода

```
logs | ParDo([] (const TQueryLogEntry& logEntry) {
    auto category = ResolveCategory(logEntry.query());
    return TKV<int, std::string>{logEntry.user_id(), category};
});

...
auto Apply(const TCollection<T>& pCollection, const F& f) {
    if constexpr (IsAppliable<F, T>) {
        // Этот указатель будет отправлен на сервер, как число.
        // Так можно передать на сервер функцию и ничего не захватывающую лямбду.
        R (*)(const T&) fPtr = f;
        ...
    }
}
```

Локализация кода

```
struct TCategoryResolver : public ITransform {
    TCategoryResolver();
    TCategoryResolver(...);
    void Do(const std::string& in, TOutput<std::string>& out) override;
    void Save(IOutputStream& output) const override;
    void Load(IInputStream& input) override;
};
```

Локализация кода

```
struct TCategoryResolver : public ITransform {
    TCategoryResolver();
    TCategoryResolver(...);
    void Do(const std::string& in, TOutput<std::string>& out) override;
    void Save(IOutputStream& output) const override;
    void Load(IInputStream& input) override;
};
...
template <typename TTransformImpl>
std::unique_ptr<ITransform> Create() {
    return std::unique_ptr<TTransformImpl>();
}
...
std::unique_ptr<ITransform> (*fPtr)() = &Create<TCategoryResolver>;
```

Исполнение кода

```
// Выполняет Map, используя переданную пользовательскую функцию
void RunMap() {
    ...
    uint64_t userFunctionPtr = ...;
    ...

}
```

Исполнение кода

```
// Выполняет Map, используя переданную пользовательскую функцию
void RunMap() {
    ...
    uint64_t userFunctionPtr = ...;
    ...
    // Напомните, какая сигнатура была у userFunction?
}
```

Исполнение кода

```
struct IRawOutput {
   virtual void Do(const void* row) = 0;
};
struct IRawParDo : public IRawOutput {
   virtual void Start(const std::vector<IRawOutput*>& outputs) = 0;
   virtual void Do(const void* row) = 0;
};
template <typename TInput, typename TOutput>
struct TRawParDoImpl : public IRawParDo {
   void Do(const void* row) override {
       T result = Function_(*static_cast<const TInput*>(row));
       Outputs_.Add(&result);
   }
   TOutput (*Function_) (const TInput&); // Передается в Save/Load как число.
   std::vector<IRawOutput*> Outputs_; // Заполняется в Start().
};
```

Исполнение кода

```
struct IRawOutput {
    virtual void Do(const void* row) = 0;
};
struct IRawParDo : public IRawOutput {
    virtual void Start(const std::vector<IRawOutput*>& outputs) = 0;
    virtual void Do(const void* row) = 0;
};
template <typename TInput, typename TOutput>
struct TRawParDoImpl : public IRawParDo {
    void Do(const void* row) override {
        T result = Function_(*static_cast<const TInput*>(row));
        Outputs_.Add(&result);
    }
    TOutput (*Function_) (const TInput&); // Передается в Save/Load как число.
    std::vector<IRawOutput*> Outputs_; // Заполняется в Start().
};
```

Исполнение кода

```
struct IRawOutput {
   virtual void Do(const void* row) = 0;
};
struct IRawParDo : public IRawOutput {
   virtual void Start(const std::vector<IRawOutput*>& outputs) = 0;
   virtual void Do(const void* row) = 0;
};
// 1. Не нужно помнить исходную сигнатуру функций.
// 2. Можно склеивать пользовательские функции в рамках оптимизаций (Unpack() | Split()).
```

Исполнение кода

```
// Выполняет Map, используя переданную пользовательскую функцию
void RunMap() {
    ...
    auto rawParDo = ...;
    rawParDo->Do(
    // Здесь нам нужен void* на прочитанный из YTsaurus объект.
    // Но сначала его надо прочитать и десериализовать.
    ...

}
```

Исполнение кода

```
struct TRowVtable {
    using TUniDataFunction = void (*)(void*);
    using TDeserializeFunction = void (*)(std::string_view, void *);
    using TSerializeFunction = std::string (*)(const void *);

    size_t DataSize = 0;
    TUniDataFunction DefaultConstructor = nullptr;
    TUniDataFunction Destructor = nullptr;
    TDeserializeFunction Deserialize = nullptr;
    TSerializeFunction Serialize = nullptr;
    // Передаём эту структуру на сервер как набор целых чисел.
    ...
};
```

Исполнение кода

```
template <typename T> void Deserialize(std::string_view, T*);

template <typename T>
TRowVTable MakeRowVTable() {
    TRowVtable vtable;
    vtable.DataSize = sizeof(T);
    vtable.DefaultConstructor = [] (void* data) {
        new(data) T;
    };
    vtable.Destructor = [] (void* data) {
        T* d = reinterpret_cast<T*>(data);
        d->~T();
    };
    vtable.Deserialize = &Deserialize<T>;
    return vtable;
}
```

Исполнение кода

```
void RunMap() {
    auto rawParDo = ...;
    auto inputRowVtable = ...;
    auto outputRowVtable = ...;

    std::vector<char> rawRow(inputRowVtable.DataSize);
    inputRowVtable.DefaultConstructor(rawRow.data());
    rawParDo->Start(std::vector{CreateMapReduceWriter(outputRowVtable)});

    for (std::string_view serializedRow : ReadInputData()) {
        inputRowVtable.Deserialize(rawRow.data());
        rawParDo->Do(rawRow.data());
    }

    inputRowVtable.Destructor(rawRow.data());
}
```

Исполнение кода

```
void RunMap() {
    auto rawParDo = ...;        // Получаем из окружения функцию с бизнес-логикой
    auto inputRowVtable = ...; // и RowVtable для входного и выходного типов
    auto outputRowVtable = ...;

    std::vector<char> rawRow(inputRowVtable.DataSize);
    inputRowVtable.DefaultConstructor(rawRow.data());
    rawParDo->Start(std::vector{CreateMapReduceWriter(outputRowVtable)});

    for (std::string_view serializedRow : ReadInputData()) {
        inputRowVtable.Deserialize(rawRow.data());
        rawParDo->Do(rawRow.data());
    }

    inputRowVtable.Destructor(rawRow.data());
}
```

Исполнение кода

```
void RunMap() {
    auto rawParDo = ...;
    auto inputRowVtable = ...;
    auto outputRowVtable = ...;

    std::vector<char> rawRow(inputRowVtable.DataSize); // теперь в rawRow лежит объект
    inputRowVtable.DefaultConstructor(rawRow.data());  //                 входного типа
    rawParDo->Start(std::vector{CreateMapReduceWriter(outputRowVtable)});

    for (std::string_view serializedRow : ReadInputData()) {
        inputRowVtable.Deserialize(rawRow.data());
        rawParDo->Do(rawRow.data());
    }

    inputRowVtable.Destructor(rawRow.data());
}
```

Исполнение кода

```
void RunMap() {
    auto rawParDo = ...;
    auto inputRowVtable = ...;
    auto outputRowVtable = ...;

    std::vector<char> rawRow(inputRowVtable.DataSize);
    inputRowVtable.DefaultConstructor(rawRow.data());
    rawParDo->Start(std::vector{CreateMapReduceWriter(outputRowVtable.Serialize)});
    // ^^^ используя сериализатор выходного типа создаём RawOutput
    for (std::string_view serializedRow : ReadInputData()) {
        inputRowVtable.Deserialize(rawRow.data());
        rawParDo->Do(rawRow.data());
    }

    inputRowVtable.Destructor(rawRow.data());
}
```

Исполнение кода

```
void RunMap() {
    auto rawParDo = ...;
    auto inputRowVtable = ...;
    auto outputRowVtable = ...;

    std::vector<char> rawRow(inputRowVtable.DataSize);
    inputRowVtable.DefaultConstructor(rawRow.data());
    rawParDo->Start(std::vector{CreateMapReduceWriter(outputRowVtable)});

    for (std::string_view serializedRow : ReadInputData()) {
        inputRowVtable.Deserialize(rawRow.data());
        rawParDo->Do(rawRow.data());
    }

    inputRowVtable.Destructor(rawRow.data());
}
```

Исполнение кода

```
void RunMap() {
    auto rawParDo = ...;
    auto inputRowVtable = ...;
    auto outputRowVtable = ...;

    std::vector<char> rawRow(inputRowVtable.DataSize);
    inputRowVtable.DefaultConstructor(rawRow.data());
    rawParDo->Start(std::vector{CreateMapReduceWriter(outputRowVtable)});

    for (std::string_view serializedRow : ReadInputData()) {
        inputRowVtable.Deserialize(rawRow.data());
        rawParDo->Do(rawRow.data());
    }

    inputRowVtable.Destructor(rawRow.data()); // не забываем почистить наш объект
}
```

YTsaurus

https://github.com/ytsaurus/ytsaurus/tree/main/yt/cpp/roren

https://github.com/ytsaurus/ytsaurus

Вопросы?

Дмитрий Ермолов,
Яндекс